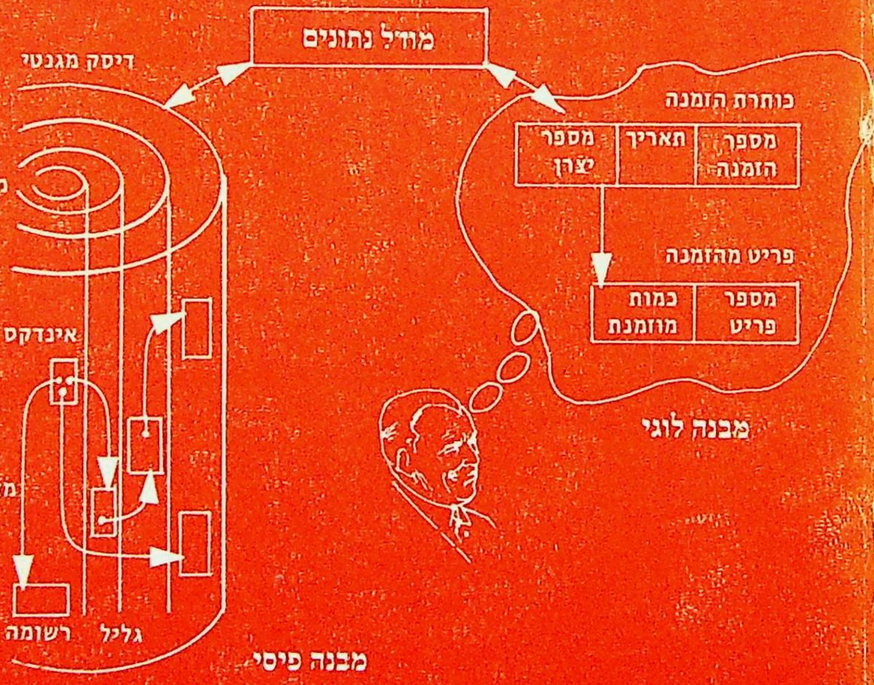
ארגון נתונים וניהול קבצים

מאת רז הייפרמן



ערך יצחק עמיהוד

ספרי לימוד והכשרה במדעי המחשב

הוצאת הוד־עמי בשיתוף המכון לפריון העבודה והייצור

ארגון נתונים
וניהול קבצים

ארגון נתונים וניהול קבצים

מאת רז הייפרמן

ערך יצחק עמיהוד

ספרי לימוד והכשרה במדעי המחשב
הוצאת הוד־עמי בשיתוף המכון לפריון העבודה והייצור

DATA ORGANIZATION and FILE MANAGEMENT

By Raz Heiferman

edited by Itzhak Amihud

הודפס בישראל
אב תשמ"ב, אוגוסט 1982
ISRAEL
August, 1982

עריכה גרפית והבאה לדפוס:
חיים אלבוחר

סדר: טכנוסדר בע"מ

מוקדש לרעייתי ארנונה
וילדי אייל ואילת

הקדמה

מערכות לניהול נתונים התפתחו בקצב מהיר בשנים האחרונות והן מהוות את אחד המרכיבים החשובים ביותר במערכות המידע הממוחשבות. קיים גידול מתמיד במספר המערכות לניהול נתונים ובמגוון מבני הנתונים שמערכות אלה מעמידות לרשות המשתמש. מטרת הספר להקנות לאנשים העוסקים בפיתוח והקמת מערכות מידע ממוחשבות בסיס להבנת עקרונות הפעולה ומבני הנתונים במערכות לניהול נתונים. הבנת מערכות אלה תאפשר למשתמש לפתור את בעית ניהול הנתונים בצורה טובה ויעילה יותר.

הספר מחולק לאחד־עשר פרקים. בפרק הראשון מוצגים מושגי היסוד של מערכות לניהול נתונים ומושם דגש על ההבדל בין המבנה הלוגי של הנתונים לבין ייצוגם הפיסי במחשב. הפרק השני מציג את היחידות השונות לאיחסון נתונים ומוסבר ההבדל בין הזכרון הפנימי של המחשב לבין היחידות החיצוניות לאיחסון נתונים. הפרק השלישי מציג את השיקולים השונים בתכנון הרשומה, שהיא יחידת המידע המשמשת לתקשורת בין המשתמש לבין המערכת לניהול נתונים. פרקים ד' עד ט' מציגים את מבני הנתונים השונים הקיימים במערכות לניהול נתונים. הפרק העשירי מציג מודל כללי לתכנון קבצים במערכות מידע ממוחשבות ואת המאפינים השונים של הקבצים. הבנת מאפינים אלה מאפשרת בחירת מבנה הנתונים המתאים ביותר לפתרון בעית עיבוד הנתונים. הפרק האחד־עשר מציג את עקרונות הפעולה של המערכות לניהול נתונים והשיטות השונות לניהול הקבצים ביחידות האיחסון. את הספר חותמים ביבליוגרפיה ומילון נתונים.

בסיום כל פרק הובאו מספר שאלות חזרה ותרגול של החומר הנלמד.

כדי להקל על הלומד לא ניתן הסבר מפורט לנוסחאות ולשיטות חישוב מסוימות בגוף החומר אלא בנספחים שבסוף כל פרק.

הספר מניח שלקורא נסיון מסוים בתכנון ויישום מערכות מידע ממוחשבות. רוב הדוגמאות מבוססות על שפת COBOL. השתדלתי, עד כמה שניתן, להתרכז בצד העקרוני של ניהול הנתונים ולא ביישום מערכת ניהול נתונים במחשב מסוים. המשתמש המעונין בהבנת יישום מערכת ניהול נתונים מסוימת מופנה לספרים הטכניים של היצרנים.

תקוותי שספר זה ימלא את החסר בספרות בשפה העברית בתחום זה, ובכך יתרום לשיפור השימוש במערכות לניהול נתונים.

תוכן העניינים

פרק רביעי: שיטת ארגון סדרתי

פרק חמישי: ארגון אינדקס סדרתי

פרק שישי: ארגון נתונים ישיר

פרק שביעי: אינדקסים משניים

פרק שמיני: ארגון נתונים מורכב — שרשרות

פרק ראשון
מבוא ומושגי יסוד בניהול נתונים

1.1 מבוא
1.2 מושגי יסוד בניהול נתונים
1.3 סיכום
1.4 שאלות

בפרק זה מוצגים מושגי היסוד במערכות לניהול נתונים.

1.1 מבוא

ארגון וניהול נתונים לשם אבטחת גישה נוחה ומהירה לאחסנה, לעדכון ולאיחזור הינם בין הנושאים המרכזיים המעסיקים א.ז המשתמשים במערכות מידע ממוחשבות. כלומר, **מערכת ניהול הנתונים הינה הבסיס לפיתוח מערכת המידע.**

ההגדרה המקובלת היא **שהמידע הינו התוצאה של עיבוד או ניתוח נתונים. הנתונים** הם חומר הגלם לתהליך זה ומורכבים מאוסף של סמלים המיצגים עובדות. הבעיה מתעוררת כאשר נתונים למשתמש אחד הם מידע עבור המשתמש השני ולהיפך. לדוגמא, קופאי המסכם בסוף היום את התקבולים באותו יום מאמין שהוא מייצר מידע חשוב אודות פעילותו במשך היום. מבחינת הארגון, לטווח הארוך אלה הם נתונים גולמיים המשמשים להפקת מידע על תחזית המכירות. המסקנה היא שתהליך תרגום הנתונים למידע הינו תהליך סוביקטיבי התלוי במטרה למענה מבוצע התרגום ובזמן התרגום. ניתן לומר שהמידע נבדל מהנתונים בכך שהוא שותף בתהליכים ארגוניים של קבלת החלטות ופתרון בעיות.

למידע יש ערך אשר הולך ופוחת עם הזמן כאשר המידע מאבד את עדכניותו. ככל שתהליך תרגום הנתונים למידע ארוך יותר, ערך המידע המופק הינו קטן יותר. המערכות לניהול נתונים הינם הכלי המאפשר בנית מסגרת ברורה לארגון הנתונים וליעול תהליך תרגומם למידע. מערכות אלה מאפשרות קבלת המידע בעיתוי מוקדם יותר, עיתוי שבו ערך המידע בשיאו.

מערכת המידע בנויה מהרכיבים הבאים:

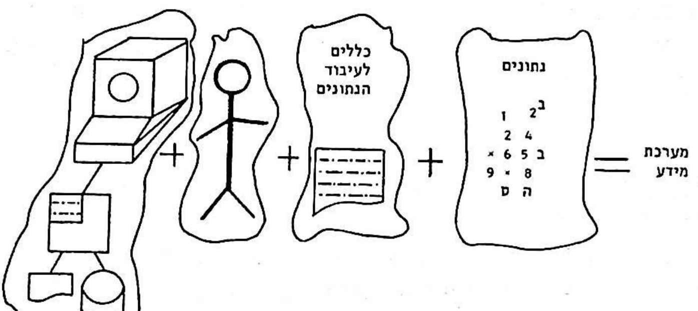
א. **אנשים הזקוקים למידע ומשתמשים בו למטרות שונות.** הארגון מורכב מאנשים המבצעים משימות מוגדרות (לדוגמא מחלקת השיווק האחראית על נושא השיווק בארגון, או מחלקת אחסנה האחראית על ניהול המלאי). מערכות המידע צריכות לשרת את הארגון בתחומים הבאים: תפעול שוטף, עזרה בתהליך קבלת החלטות, פתרון בעיות, ניתוח נתונים ועוד.

ב. **כללים לעיבוד הנתונים.** זהו אוסף של כללים המגדיר את תהליכי העיבוד ואת צורת תרגום הנתונים למידע. כללים אלה תלויים בסביבה בה פועל הארגון. לדוגמא, כללי הנהלת חשבונות יקבעו מהם האירועים שיש להזין בהם את מערכת המידע.

ג. **מכונות לעיבוד נתונים המבצעות את תהליך התרגום מנתונים למידע.** מכונות אלה יכולות להיות מחשבים, מכונות להנהלת חשבונות, קופות רושמות ועוד. האדם עצמו שותף גם הוא בתהליך התרגום. מערכות המידע הממוחשבות מנצלות את המהירות העצומה של המחשב לעיבוד כמויות גדולות של נתונים, בדייקנות ובצורה אחידה.

ד. **נתונים, שהם "לב" מערכת המידע.** הנתונים מאוחסנים ביחידות איחסון חיצוניות למחשב, מאחר וקימת מגבלה של כמות הנתונים שניתן לאחסן בזכרון המחשב (אחסנה פנימית). אם עלות האיחסון של הנתונים עולה על ערך המידע המופק מהם, לא נהיה מעונינים לאחסן אותם. ככל שערך המידע המופק הולך וקטן נהיה מעונינים להעביר נתונים אלה ליחידות איחסון זולות יותר, עד כדי אי־שמירה.

התכנון והגדרת המטרות של מערכת מידע חייבים להתחשב בכל הרכיבים האלה ולראותם כשלימות אחת. התחשבות רק בטכנולוגית המידע (שיטות העיבוד במחשב, שיטות ארגון קבצים, שיטות אחסון) יכולה להביא לכשלון המערכת. קביעת המטרות של מערכת המידע והגדרת הדרישות הם עיקר תפקידיו של מנתח המערכות. למרות שבספר זה נתרכז בעיקר בהיבטים שונים של טכנולוגית המידע, על מנתח המערכות להבין שזהו רק אחד מתוך המכלול הרחב של היבטים של תהליך **ניהול וארגון הנתונים במערכת המידע.**



תרשים 1-1: מרכיבי מערכת מידע

ניתן לחלק את מערכות עיבוד הנתונים לשני מרכיבים עיקריים:

— **אוסף של תוכניות מחשב המיועדות לפתרון בעית עיבוד הנתונים.** תוכניות אלה מיצגות את הכללים לעיבוד הנתונים (האלגוריתמים) והן נכתבות על־ידי המשתמש בשפה עילית כלשהי, כמו קובול או PL/1. התוכניות משמשות לניסוח ותרגום הבעיה לשפה המובנת על־ידי המחשב.

— **אוסף הנתונים הדרושים לפתרון הבעיה.** נתונים אלה מנוהלים בעזרת אוסף של תוכניות מחשב מיוחדות לפתרון בעית ניהול הנתונים (DMS או Data Management Systems). תכניות אלה נכתבות בדרך כלל על־ידי יצרן המחשב או בתי תוכנה המתמחים בסוג זה של תוכניות. מורכבותם הרבה של תוכניות אלה והצורך בניצול יעיל של משאבי המחשב מחייבת את כתיבתם בשפת סף (Assembly Language) מיוחדת לכל מחשב.

אי־תלות פיסית (Physical Data Independence)

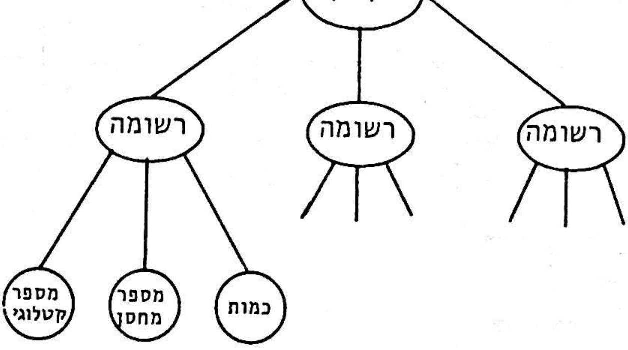
המבנה הלוגי של הנתונים הינו הצורה בה רואה המשתמש את הנתונים. צורה זו יכולה להיות שונה מאד מהצורה בה הנתונים מאוחסנים ביחידות האחסון. המבנה הלוגי האידיאלי הוא המבנה המאפשר נוחות שימוש מירבית למשתמש. ככל שהמבנה הפיסי שקוף יותר מבחינת המשתמש נאמר שקימת **אי־תלות פיסית** גדולה יותר. להשגת אי־תלות יש לבנות **מודל נתונים** המאפשר את ביצוע התרגום בין המבנה הפיסי של הנתונים למבנה הלוגי.



תרשים 1-2: מודל הנתונים מתרגם את המבנה הפיסי למבנה לוגי

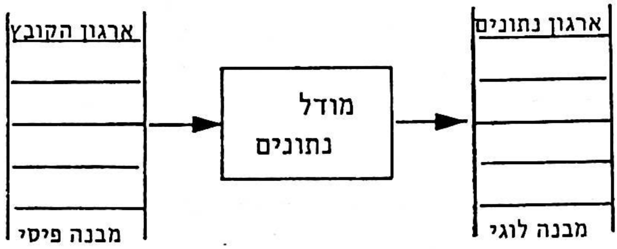
במקום לפנות לנתונים על סמך התכונות הפיסיות שלהם (כגון מספר דיסק, מספר גליל, מספר מסילה, מספר הבתים וכדומה) בונים מודל נתונים שהינו **אוסף כללים** לייצוג הנתונים והקשרים ביניהם. המודל אינו מתיחס לנתונים עצמם המאוחסנים ביחידות האחסון, כי אם לדרך בה הם **מאורגנים מבחינה לוגית.**

לדוגמא נסתכל על יישום של ניהול מלאי. עבור כל פריט במלאי יש לנהל את הנתונים הבאים: מספר קטלוגי, מספר מחסן, כמות במלאי. לכל פריט במלאי ננהל את הנתונים האלה במסגרת רשומת מלאי. אוסף כל רשומות המלאי יוגדר כקובץ המלאי. תרשים 1-3 מציג את המבנה הלוגי שבנינו. ניתן לראות שבמודל לא התיחסנו כלל לתוכן הנתונים כפי שהם מופיעים ביחידת האיחסון אלא רק למבנה הלוגי שלהם.



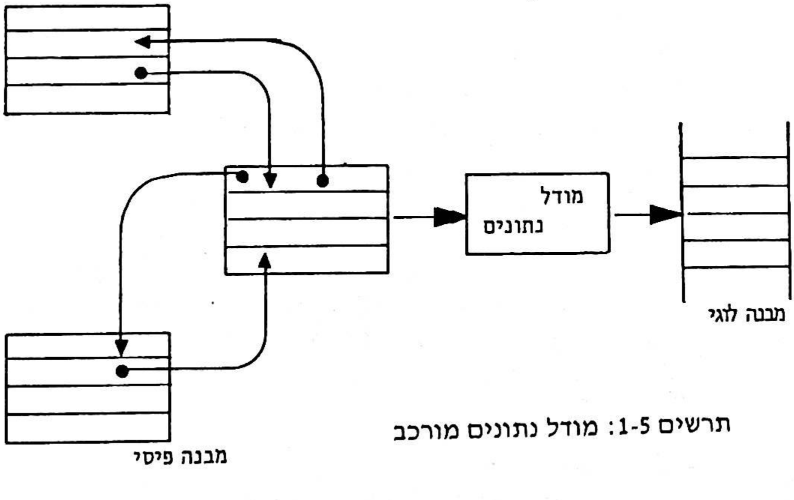
תרשים 1-3: מודל לוגי קובץ מלאי

כל רשומה לוגית ניתן לאחסן ברשומה פיסית אחת. זהו מודל הנתונים הפשוט ביותר והוא מיושם ברוב המערכות לניהול קבצים.



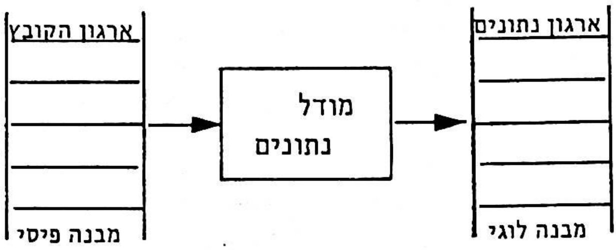
תרשים 1-4: מודל נתונים פשוט

כפי שנראה בהמשך קיימות שיטות לארגון הקובץ בהן המשתמש פונה לרשומה על־ידי ציון המפתח של הרשומה המבוקשת. המשתמש אינו יודע כלל את הדרך בה המערכת לניהול קבצים מאתרת את הרשומה באופן פיסי. ארגון מחדש של הקובץ יכול לגרום לתזוזה של רשומות ממקום אחד למקום אחר אבל אין לכך כל השפעה על המבנה הלוגי שאותו רואה המשתמש. עם זאת אין אפשרות לומר שקימת כאן אי־תלות פיסית מוחלטת מאחר והמשתמש חייב בכל זאת להכיר את שיטת הארגון של הקובץ ולהשתמש בכללים המיוחדים לשיטה. עליו להכיר גם את המבנה המדויק של הרשומות בקובץ כדי שיוכל לטפל בהן. למערכות המשתמשות במודל נתונים זה נקרא **מערכות לניהול קבצים** (FMS — File Management Systems). מערכות אלו מציגות בפני המשתמש מספר שיטות מצומצם יחסית לארגון הנתונים. שיטות אלה מוצגות בפרקים 4, 5, 6, 7.



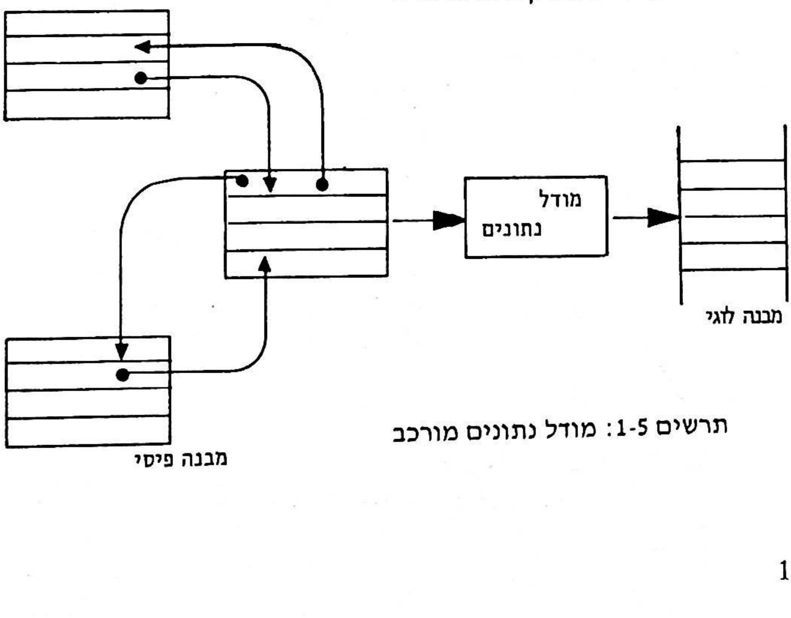
תרשים 1-5: מודל נתונים מורכב

כל רשומה לוגית ניתן לאחסן ברשומה פיסית אחת. זהו מודל הנתונים הפשוט ביותר והוא מיושם ברוב המערכות לניהול קבצים.



תרשים 1-4: מודל נתונים פשוט

כפי שנראה בהמשך קיימות שיטות לארגון הקובץ בהן המשתמש פונה לרשומה על־ידי ציון המפתח של הרשומה המבוקשת. המשתמש אינו יודע כלל את הדרך בה המערכת לניהול קבצים מאתרת את הרשומה באופן פיסי. ארגון מחדש של הקובץ יכול לגרום לתזוזה של רשומות ממקום אחד למקום אחר אבל אין לכך כל השפעה על המבנה הלוגי שאותו רואה המשתמש. עם זאת אין אפשרות לומר שקימת כאן אי־תלות פיסית מוחלטת מאחר והמשתמש חייב בכל זאת להכיר את שיטת הארגון של הקובץ ולהשתמש בכללים המיוחדים לשיטה. עליו להכיר גם את המבנה המדויק של הרשומות בקובץ כדי שיוכל לטפל בהן. למערכות המשתמשות במודל נתונים זה נקרא **מערכות לניהול קבצים** (FMS — File Management Systems). מערכות אלו מציגות בפני המשתמש מספר שיטות מצומצם יחסית לארגון הנתונים. שיטות אלה מוצגות בפרקים 4, 5, 6, 7.



תרשים 1-5: מודל נתונים מורכב

אי־תלות פיסית (Physical Data Independence)

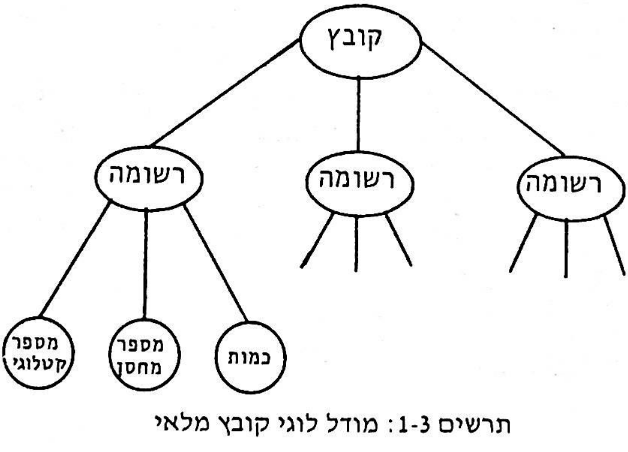
המבנה הלוגי של הנתונים הינו הצורה בה רואה המשתמש את הנתונים. צורה זו יכולה להיות שונה מאד מהצורה בה הנתונים מאוחסנים ביחידות האחסון. המבנה הלוגי האידיאלי הוא המבנה המאפשר נוחות שימוש מירבית למשתמש. ככל שהמבנה הפיסי שקוף יותר מבחינת המשתמש נאמר שקימת **אי־תלות פיסית** גדולה יותר. להשגת אי־תלות יש לבנות **מודל נתונים** המאפשר את ביצוע התרגום בין המבנה הפיסי של הנתונים למבנה הלוגי.



תרשים 1-2: מודל הנתונים מתרגם את המבנה הפיסי למבנה לוגי

במקום לפנות לנתונים על סמך התכונות הפיסיות שלהם (כגון מספר דיסק, מספר גליל, מספר מסילה, מספר הבתים וכדומה) בונים מודל נתונים שהינו **אוסף כללים** לייצוג הנתונים והקשרים ביניהם. המודל אינו מתיחס לנתונים עצמם המאוחסנים ביחידות האחסון, כי אם לדרך בה הם **מאורגנים מבחינה לוגית.**

לדוגמא נסתכל על יישום של ניהול מלאי. עבור כל פריט במלאי יש לנהל את הנתונים הבאים: מספר קטלוגי, מספר מחסן, כמות במלאי. לכל פריט במלאי ננהל את הנתונים האלה במסגרת רשומת מלאי. אוסף כל רשומות המלאי יוגדר כקובץ המלאי. תרשים 1-3 מציג את המבנה הלוגי שבנינו. ניתן לראות שבמודל לא התיחסנו כלל לתוכן הנתונים כפי שהם מופיעים ביחידת האיחסון אלא רק למבנה הלוגי שלהם.



תרשים 1-3: מודל לוגי קובץ מלאי

קיימת אפשרות לארגן את המבנה הפיסי של הנתונים בצורה שונה לחלוטין מהמבנה הלוגי. בתרשים 1-5 הנתונים קשורים ביניהם בעזרת מצביעים שונים. המשתמש אינו צריך להכיר את הארגון הפיסי של הנתונים. הוא ממשיך לראות לפניו מבנה לוגי כלשהו ורק בו הוא מטפל. המשתמש יכול לפנות לנתונים במושגים "הרשומה הבאה" או "הרשומה הקודמת" מבלי להתיחס כלל **למצביעים** (Pointers) הקושרים קטעי רשומות פיסיות לרשומה לוגית אחת. המערכות המיישמות מודלים מורכבים יותר של נתונים הן **המערכות לניהול בסיסי נתונים** (DBMS — Data Base Management Systems). מערכות אלה משתמשות במבני נתונים פיסיים מורכבים מאד (מבנים אלה מוצגים בפרק 8 ו־9) ומאפשרות למשתמש אי־תלות פיסית.

היתרונות של אי־תלות פיסית הם:

א. **נוחות שימוש מנקודת מבט המשתמש.** המודל הלוגי בלתי תלוי בצורת ההצגה הפיסית ומשחרר את המשתמש מלהתיחס לבעיות הטכניות של אחסון הנתונים. תהליך פיתוח תוכנת היישומים פשוט יותר.

ב. **אפשרות לבצע שינויים במבנה הפיסי של הנתונים ללא השפעה על תוכניות היישום.** לדוגמא ניתן להשתמש בטכנולוגיות אחסון נתונים חדשות (כגון דיסק חדש) ללא צורך בביצוע שינויים כלשהם בתוכניות.

כאן המקום לציין שקיימת סכנה מסוימת באי־תלות פיסית. מאחר והמשתמש אינו מכיר את המבנה הפיסי של הנתונים ואת שיטת אירגונם הוא יכול להשתמש במערכת בצורה לא יעילה (לדוגמא, אם המשתמש אינו יודע שגישה לרשומה לפי מפתח מחייבת סריקות נרחבות של כל הקובץ).

המערכות לניהול נתונים מטפלות במגוון רחב של פונקציות. לדוגמא: הגדרת הנתונים, כתיבה ביחידות אחסון וארגונם כדי שניתן יהיה לשלוף אותם ביעילות. כמו כן מבצעות מערכות אלה מספר תפקידים שהמשתמש אינו מכיר והם "שקופים" מבחינתו. לדוגמא: ניהול מאגרים (Buffers), ניהול אינדקסים, חישובי כתובות ועוד. באופן עקרוני רוב המערכות לניהול נתונים מכילות שני מרכיבים עיקריים:

א. **שפה להגדרת נתונים** (DDL — Data Definition Language) שבעזרתה מגדירים את הנתונים בצורה נוחה למשתמש.

ב. **שפה לטיפול בנתונים** (DML — Data Manipulation Language) המאפשרת את ביצוע הפעולות השונות כגון שליפה, איחסון וכדומה.

נציג כאן קטע של תוכנית קובול להדגמת שני מרכיבים אלה. נציין רק שקובול אינו מערכת לניהול נתונים כי אם שפת תכנות המשתמשת בשירותים של המערכת לניהול נתונים. ניהול הנתונים מבוצע על ידי התוכנה לניהול נתונים מחוץ לתוכנית הקובול.

```
SELECT MLAY-FILE ASSIGN TO XX
       ORGANIZATION IS SEQUENTIAL.
= = =
FD       MLAY-FILE
         LABEL RECORD IS OMITTED
         DATA RECORD IS MLAY-REC.
01       MLAY-REC.
       03 MISP-QATALOGI          PIC 9(9).
       03 MAHSAN                 PIC 9(3).
       03 KAMUT                  PIC 9(8) V99.

PROCEDURE DIVISION.
= = =
OPEN   INPUT MLAY-FILE
       OUTPUT MLAY-OUT.
= = =
READ   MLAY-FILE AT END PERFORM SOF.
PERFORM TIPUL
WRITE MLAY-OUT FROM MLAY-REC.
= = =
CLOSE MLAY-FILE MLAY-OUT.
= = =
```

DDL
הגדרת נתונים

DML
טיפול בנתונים

כפי שניתן לראות מחולקת התוכנית לשני קטעים: קטע להגדרת הנתונים וקטע לטיפול בהם. בקטע להגדרת נתונים מגדירים את מבנה הרשומה בעזרת שמות סימליים הניתנים הן לרשומה והן לשדות המרכיבים אותה. בקטע לטיפול בנתונים משתמשים בפקודות DML כדי לקרוא ולאחסן את הנתונים ובפקודות קובול לטיפול ברשומות. כפי שניתן לראות מתוך התוכנית, השיטה בה מתבצעת הגישה לנתונים לשליפת הרשומה שקופה לחלוטין מבחינת המשתמש. פניתו לנתונים מתבצעת ברמה הסימלית בלבד.

בנוסף לבעיות האחסון ועדכון הנתונים, שהן בעיות חשובות מאד וידונו בהרחבה בספר, נרחיב במקצת את הדיון בבעיית שליפת המידע. שתי השאלות הבסיסיות בכל מערכת המאחסנת נתונים הן:

— היכן הנתון שאנו מחפשים?
— היכן הנתון "הבא" העונה על קריטריונים מסוימים?

להבהרת שאלות אלה נשתמש בדוגמא הלקוחה ממערכת ידנית לניהול ספריות. אחסון הספרים בספריה יכול להתבצע לפי מספר שיטות. לדוגמא:

— לפי מספר קטלוגי (בעל מבנה לוגי מסוים),
— לפי נושא,
— לפי שם ספר (אלפה־ביתי),
— לפי סדר קבלה.

על מנת לאתר את הספר בעת הצורך, יש להכיר את שיטת אחסון הספר מכיון שהיא קובעת את שיטת איתורו. נסקור עתה את התהליך המחשבתי של האיתור:

א. **היכן הספר:** לאיתור ספר בספריה יש להכיר את שיטת האחסון, מכיון שהיא קובעת את מקום הספר וגם את שיטת איתורו. בחירת שיטת אחסון הינה בעלת חשיבות רבה מאחר ובחירת שיטה לא נכונה יכולה להכביד מאד על השימוש בספריה. החלפת שיטת האחסון תחייב שינוי בשיטת איתור הספר. מקום אחסון הספר יהיה שונה אם נאחסן אותו לפי מספר קטלוגי או לפי סדר קבלה.

ב. **מהו הספר הבא:** ברגע שמאתרים ספר מסוים, הספר "הבא" נקבע גם הוא על־ידי שיטת האחסון. לדוגמא נניח ששיטת האחסון היא לפי נושא. כאשר מאתרים ספר מסוים בנושא "ארגון קבצים" הרי שהספר המאוחסן לידו יהיה גם הוא באותו נושא, אם כי יתכן שנכתב על־ידי מחבר שונה. שיטת האיחסון הפיזית של הספרים קובעת את הסדר הלוגי שלהם ועונה על השאלה "מהו הספר הבא".

הבעיה מתעוררת כאשר קריטריון החיפוש או האיתור אינו זהה לקריטריון האחסון. כאשר רוצים לשלוף ספר של מחבר מסוים או לקבל את רשימת כל הספרים שחוברו על ידו, לא ניתן לעשות זאת אם שיטת אחסון הספרים אינה מאפשרת שליפת ספרים לפי קריטריונים אלה. מאחר והספרים מאוחסנים לפי נושאים, יתכן שכל הספרים של מחבר מסוים "מפוזרים" על פני כל הספריה ולכן יש לסרוק את כל הספרים בספריה בכדי לאתר את הספרים המבוקשים. על מנת להתגבר על בעיה זו בונים אינדקסים, הרשומים בדרך כלל בכרטסות, והם מאפשרים לבצע תרגום בין קריטריון החיפוש לבין קריטריון האחסון. לדוגמא נבנה אינדקס הממוין לפי שם סופר כדי שאפשר יהיה בעזרתו לאתר את כל הספרים של סופר מסויים, אף־על־פי שהספרים אינם מאוחסנים בשיטה זו.

ניתן לומר שהמטרה היא לבצע הפרדה בין הסדר הפיסי שבו מאוחסנים הספרים לבין הסדר הלוגי שלהם. אפשרות אחת היא שימוש באינדקסים.

שיטה אחרת לביצוע הפרדה זו היא שימוש במצביעים (Pointers).

נדגים שימוש במצביעים מחיי היום יום ונסתכל על עמוד של עתון. רוב הכתבות בעמוד הראשון מפוצלות והמשכן מופיע בעמוד אחר כלשהו. מבחינה פיסית הכתבה מפוצלת בשני מקומות. בכדי ליצור את הרצף הלוגי לקריאת הכתבה יש להשתמש במצביע "ההמשך בעמוד 21" המגשר בין הסדר הלוגי לסדר הפיסי של קטעי הכתבה.

1.2 מושגי יסוד בניהול נתונים

בסעיף זה נבהיר את המושגים בהם נשתמש בהמשך ונגדיר אותם בצורה פורמלית.

א. **נתונים** (Data): עובדות גולמיות בלתי מעובדות. הנתונים הם ייצוג המציאות כפי שהיא באה לידי ביטוי במחשב. הנתונים הם חומר הגלם בתהליך עיבוד הנתונים.

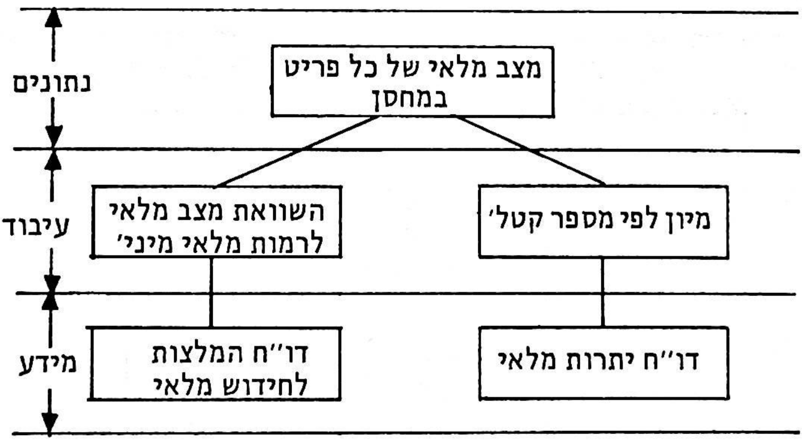
ב. **מידע** (Information): התוצר הסופי של תהליך עיבוד הנתונים הינו מידע. המידע מופק על־ידי סיווג ועיבוד הנתונים כדי שיוכלו לשמש לקבלת החלטות או לפתרון בעיות.

ג. **עיבוד נתונים** (Data Processing): תהליך ההופך נתונים למידע. העיבוד כולל סיווג, מיון, מיזוג, ביצוע חישובים, סיכומים וכדומה.



תרשים 1-6: תהליך עיבוד נתונים

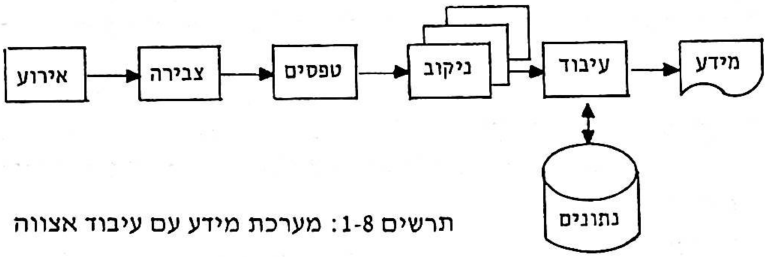
ניתן להפעיל תהליכי עיבוד שונים על אותם נתונים לקבלת מידע שונה. לדוגמא, במערכת לניהול מלאי במחסן מנהלים כרטסת (או רשומות במחשב) המכילה את הכמות במלאי לכל אחד מהפריטים. מיון הנתונים האלה לפי מספר קטלוגי יאפשר הפקת דוח יתרות מלאי לתפעול שוטף של המחסן. מצד שני השוואת הנתונים האלה מול רמות מלאי מינימום במחסן יאפשר הפקת דוח המלצות לחידוש מלאי.



תרשים 1-7: מידע שונה המופק מנתונים זהים

ד. מערכות מידע ממוחשבות (Computerized Information Systems):
מערכות המנהלות ומעבדות את הנתונים בעזרת מחשב. מערכות המידע מכילות אוסף נתונים המאוחסנים ביחידות אחסון ואוסף של תוכניות מחשב המבצעות את העיבוד ומפיקות מידע. אחד המאפיינים החשובים של מערכת מידע ממוחשבת הוא זמן התגובה שלה. מבחינים בין שני סוגי מערכות:

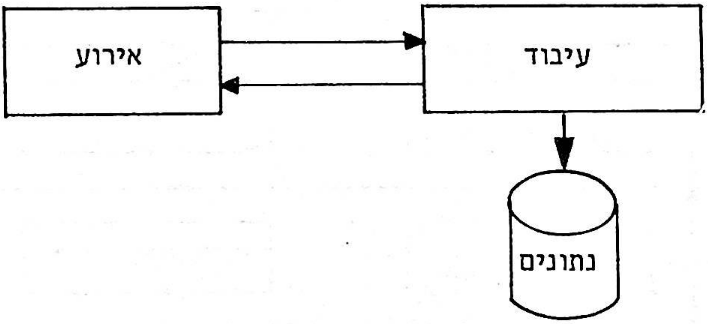
— **מערכות אצווה** (Batch Processing): מערכות הצוברות את האירועים ומעבדות את כולם במכלול אחד, ברצף.



תרשים 1-8: מערכת מידע עם עיבוד אצווה

עלות העיבוד במערכות מסוג זה נמוכה מאשר בשיטת עיבוד שנזכיר בהמשך. זמן התגובה לאירוע ארוך יחסית בגלל תהליך הצבירה. לדוגמא, צבירת תנועות ניפוק ממחסן ועיבודם פעם בשבוע יגרום לכך שנגלה רק כעבור שבוע שהפריט ירד מתחת לרמת מלאי מינימום. מסיבה זו למשל, עוברים במערכות ניהול מלאי ובעיקר במערכות בנקאיות לשיטות עיבוד אחרות, שבהן זמן התגובה קצר יותר.

— **מערכות מקוונות** (On-line Processing): מערכות המגיבות בזמן קצר ביותר לכל אירוע. האירועים מוזנים למערכת העיבוד בזמן הווצרותם באמצעות מסופים ומערכת תקשורת. מערכות אלה הן יקרות בגלל התשתית הדרושה לרשת התקשורת ואחסון הנתונים (דיסקים). לדוגמא, במערכת בנקאית, משיכת כסף מהחשבון של הלקוח או הפקדה תגרום לעדכון מיידי של היתרה. מערכות אלה מורכבות יותר בגלל זמני התגובה הקצרים וריבוי המשתמשים הניגשים באותו זמן לנתונים.



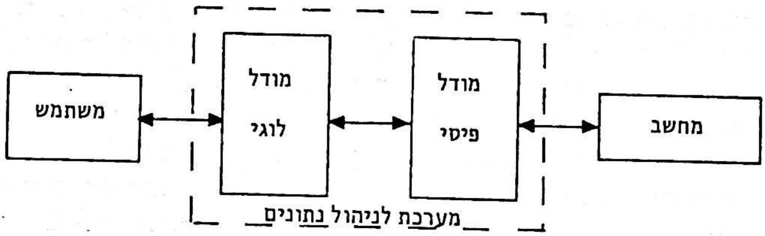
תרשים 1-9: מערכת מידע מקוונת

ה. **ניהול נתונים** (Data Management): ייצוג וניהול הנתונים במחשב נעשה על־ידי מערכת תוכנה המהווה חלק ממערכת המידע הממוחשבת. ניהול הנתונים דן בנושאים הבאים:

— הגדרת נתונים,
— ייצוג נתונים ביחידות אחסון,
— אפשרויות גישה לנתונים למטרות איחזור ועיבוד.

המערכת לניהול נתונים מכילה שני מודלים של נתונים: המודל הלוגי והמודל הפיסי. **המודל הלוגי** היא הצורה בה רואה המשתמש את הנתונים המאוחסנים במערכת המידע. **המודל הפיסי** היא הצורה בה מערכת המחשב מתיחסת לנתונים, כלומר צורת הייצוג הפנימי שלהם במחשב.

המערכת לניהול נתונים מבצעת את התרגום בין המודל הלוגי למודל הפיסי. ברוב המערכות הקיימות, המודל הפיסי שקוף מנקודת מבט המשתמש אשר רואה לפניו את המודל הלוגי בלבד.

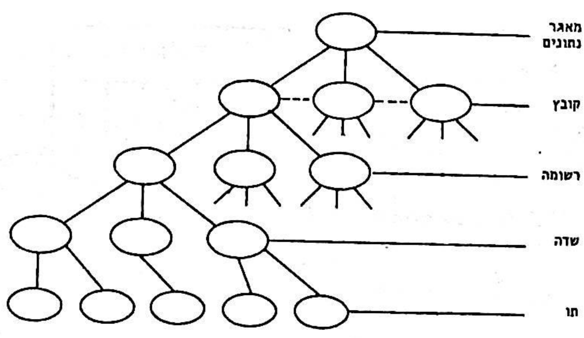


תרשים 1-10: תרגום המבנה הלוגי למבנה הפיסי

נציג את מושגי היסוד הקשורים לכל אחד משני המודלים.

— **מבנים לוגיים** (Application Oriented Data Division): מבנים אלה מטופלים על־ידי המשתמש במערכת המידע הממוחשבת. מבנים אלה קיימים רק מבחינה לוגית. צורת האחסון שלהם במחשב או ביחידת האחסון יכולה להיות שונה. הייצוג הפיסי של הנתונים אינו מעניין את המשתמש.

קיימת הירכיה בהגדרת המבנים הלוגיים:



תרשים 1-11: הירכית המבנים הלוגיים

א. **תו** (Character): יחידת הנתונים הבסיסית. מקבילה לאות, ספרה או סימן מיוחד. לדוגמא, הצירוף "AB24" מכיל 4 תווים מתוכם שתי אותיות ושתי ספרות.

ב. **שדה** (Field): אוסף תווים (אחד או יותר) בעלי משמעות לוגית. כל שדה הינו יחידת מידע. המשמעות מתקבלת רק בעקבות ההתיחסות לכל התווים המרכיבים את השדה. לשדה יש תכונות כגון גודל וסוג וכן **ערך** (Value) המייצג את הנתון.

לדוגמא, שדה המכיל מספר תעודת זהות מורכב מ־8 תווי ספרות המייצגים את מספר הזהות של העובד. שדה המכיל את כתובת העובד מורכב מ־50 תווים אלפנומריים (המכילים אותיות וספרות).

ג. **רשומה** (Record): אוסף של **שדות בעלי קשר לוגי** ביניהם. הרשומה היא יחידת המידע החשובה ביותר מבחינת המשתמש. גודל הרשומה, השדות המרכיבים אותה והמבנה הפנימי שלה תלויים ביישום.

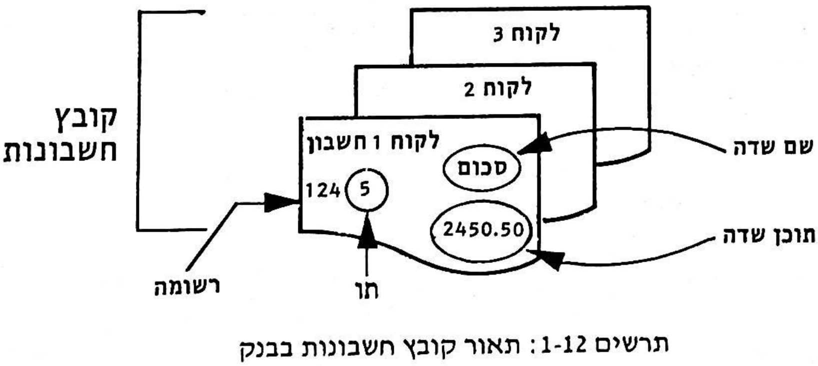
לדוגמא, ביישום משכורת מכילה הרשומה את הנתונים הרלוונטים לחישוב משכורת של עובד כלשהו אחד.

ד. **קובץ** (File): אוסף רשומות השייכות לנושא מסוים. אוסף זה הוא יחידות מידע של פרטים באוכלוסיה כלשהי. הקובץ מכיל רשומה אחת או יותר עבור כל יחידת המידע. תוכניות המחשב פונות לקבצים, שולפות מהם רשומות, מעדכנות שדות ברשומות. עדכון רשומות בקובץ משקף את השינויים שחלו באוכלוסיה המיוצגת בקובץ.

לדוגמא, אוסף כל רשומות העובד מרכיב את קובץ העובדים במפעל, ואוסף רשומות יתרות החשבון בבנק מרכיב את קובץ חשבונות הבנק.

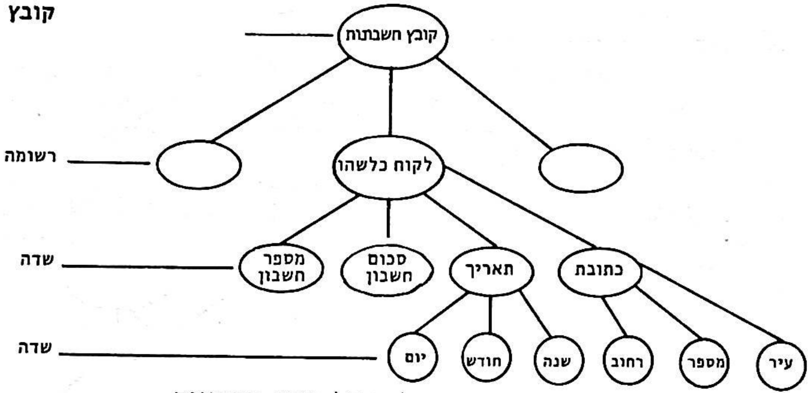
ה. **בסיס נתונים** (Data Base): אוסף כל הנתונים המנוהלים במערכת המידע. אוסף זה מכונה לעיתים **מאגר נתונים**. קיימים מאגרי נתונים המורכבים מאוסף קבצים נפרדים, בהם הקשר בין הנתונים מוגדר על־ידי התוכניות המעבדות אותם. לעומתם קיימים מאגרי נתונים בהם מיוצגים הנתונים והקשרים ביניהם כשלימות אחת.

דוגמא: קובץ חשבונות של לקוחות.



תרשים 1-12: תאור קובץ חשבונות בבנק

בדוגמא זו הקובץ מורכב מאוסף של כרטיסים, כל כ
החשבון, סכום החשבון ופרטים נוספים. מבנה זה מופי
המבנה הלוגי המקביל מופיע בתרשים 1-13.



תרשים 1-13: מבנה לוגי של קובץ חשבונות

כפי שניתן לראות קיימת כאן הירכיה מסויימת בין השדות עצמם. לדוגמא, השדה "תאריך" מתחלק לשלשה תת־שדות: יום, חודש, שנה.

הטיפול בקובץ מתבצע על־ידי המשתמש כאשר הוא רואה לפניו מבנה לוגי זה בעוד אשר המבנה הפיסי יכול להיות שונה לחלוטין. הוא תלוי ביחידת האיחסון ובהיבטים אחרים המשפיעים על יעילות העיבוד. המבנה הלוגי המתואר קיים למעשה רק במוחו של המשתמש.

— מבנים פיסיים (Machine Oriented Data Division): מבנים אלה מטופלים על־ידי המחשב ובדרך כלל הם "שקופים" מנקודת מבט המשתמש.

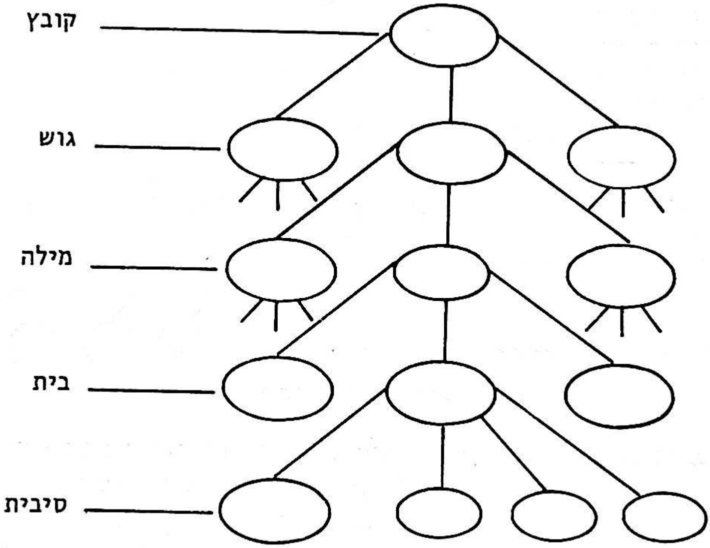
א. **סיבית** (Bit): סיפרה בינרית אשר יכולה לקבל את הערכים 0 או 1. זהו המבנה הבסיסי ביותר במחשב.

ב. **בית** (Byte): אוסף של 8 סיביות המשמשות לייצוג תו או שתי ספרות. זוהי היחידה הבסיסית המיוצגת על־ידי כתובת בזכרון המחשב (במחשבי יבמ למשל).

ג. **מילה** (Word): המילה מורכבת ממספר בתים (במחשבי יבמ — 4 בתים) ותלויה בסוג המחשב. הפנייה בזכרון מתבצעת לפי כתובת המילה במחשבים שבהם זוהי היחידה הבסיסית. במקרים אחרים יש למילה משמעות לוגית בלבד או שהיא מהווה יחידה בסיסית לעיבודים מתמטיים בלבד.

ד. **גוש** (Block): אוסף של בתים או מילים המטופלים על־ידי המחשב בפעולת קלט/פלט אחת. הגוש מכיל את הנתונים המועברים בין הזכרון ליחידות האחסון.

ה. **קובץ** (File): קבוצה של גושים השייכים לקובץ לוגי אחד. הקובץ מאוחסן ביחידת אחסון כלשהי (סרט מגנטי, דיסק מגנטי, תקליטון וכדומה). הגושים השייכים לקובץ לוגי אחד אינם חייבים להיות מאוחסנים בצורה רציפה דווקא.



תרשים 1-14: הירככית המבנים הפיסיים

1.3 סיכום

בפרק זה הצגנו את מושגי היסוד במערכות לניהול נתונים. יצרני המחשבים, מוסדות המחקר ובתי תוכנה הקדישו מאמץ רב לשיפור ויעול ניהול הנתונים. המערכות לניהול נתונים משחררות את מנתח המערכות מבעיות ייצוג הנתונים הפיסי במחשב, ומאפשרות לו להתרכז בבנית היישום ובחירת מבנה הנתונים המתאים ביותר לפתרון בעית עיבוד הנתונים. כל מתקן מחשב משתמש במערכת כלשהי לניהול נתונים. מכאן נובעת החשיבות הרבה בהבנת עקרונות פעולתן והכרת מבני הנתונים שהן מציגות בפני המשתמש.

1.4 שאלות

1. רשום אוסף של שדות נתונים שיש לדעתך לנהל במערך משכורת. ציין כיצד תארגן נתונים אלה ברשומות ובקבצים בהם תשתמש.

2. הסבר מהם הנתונים, העיבוד והמידע בתהליך חישוב משכורת באמצעות מחשב.

3. הסבר מהו ההבדל בין "שם הנתון" לבין ה"ערך" שהנתון מקבל בקובץ. הבא מספר דוגמאות.

4. אילו תכונות של המחשב גרמו לכך שהוא הפך להיות גורם רב חשיבות בהתפתחות החברה המודרנית? התיחס לחומרה ותוכנה.

5. הבא דוגמא למבנה לוגי ולצורה בה ניתן לתרגם אותו למבנה פיסי. האם יתכן מבנה פיסי אחר הנותן את אותו מבנה לוגי?

6. תן דוגמא של מערכת מידע אשר חייבת לפעול בעיבוד מקוון למרות שידוע שעלות המשאבים בסוג זה של מערכות גבוהה יותר מאשר בעיבוד אצווה.

7. מהי חשיבות אי־התלות הפיסית במערכות מידע? מדוע התפתח מושג זה?

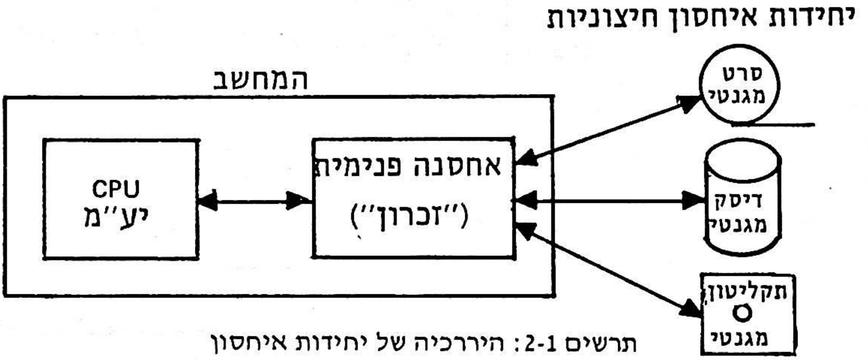
פרק שני
יחידות לאחסון נתונים

2.1 מבוא
2.2 אחסנה פנימית
2.3 מבוא ליחידות אחסון חיצוניות
2.4 סרט מגנטי
2.5 דיסק מגנטי
2.6 מאגרי קלט/פלט
2.7 שאלות
2.8 נספח א': חישוב גודל גוש בסרט מגנטי
2.9 נספח ב': חישוב זמן חיפוש בדיסק

בפרק זה יוצגו אמצעי אחסון הנתונים העיקריים ויוסברו תכונותיהם ושיטות עבודתם.

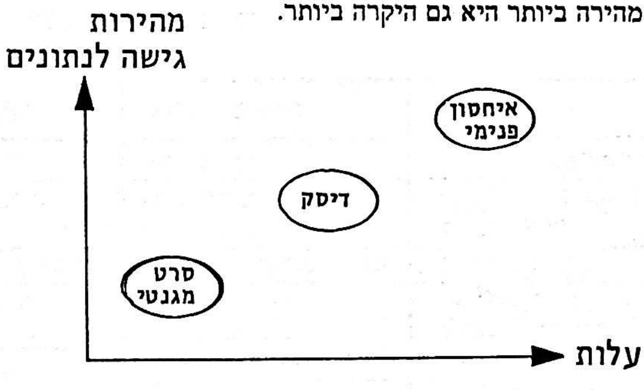
2.1 מבוא

מערכות המידע הממוחשבות מעבדות כמויות גדולות של נתונים. אחת הבעיות המרכזיות במערכות אלה היא אחסון הנתונים במחשב. אם ניתן היה לאחסן את כל הנתונים בזכרון הפנימי של המחשב לא היתה נוצרת בעית ניהול הנתונים. בגלל הכמויות העצומות של הנתונים והעלות הגבוהה לאיחסונם בזכרון, פותחו יחידות אחסון חיצוניות המאפשרות אחסון נתונים בעלות נמוכה יחסית. לכל יחידת אחסון כזאת יש יחס עלות/ביצוע מסוים וכן "גיאומטריה" משלה הקובעת את הצורה, המהירות ושיטת הגישה לנתונים.



תרשים 2-1: היררכיה של יחידות איחסון

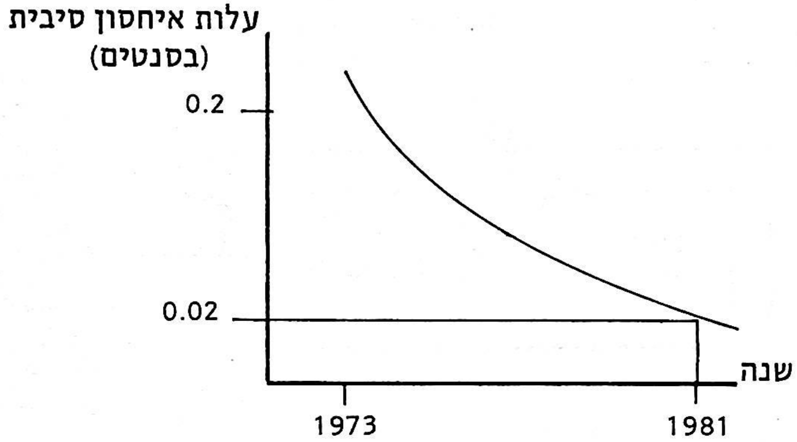
לביצועים של יחידת האחסון חשיבות רבה מאחר והם קובעים את מהירות הגישה לנתונים. השיפור בביצועים של יחידת האיחסון כרוך בהגדלת העלות לאיחסון יחידת מידע. באופן טבעי נוצרה היררכיה של יחידות איחסון, כאשר היחידה המהירה ביותר היא גם היקרה ביותר.



תרשים 2-2: מהירות יחידת האיחסון כפונקציה של עלות היחידה

בגלל היחסים השונים של עלות/ביצוע משתמשים רוב המחשבים בכמות מוגבלת של אחסנה פנימית עבור התוכניות הפועלות וחלק מהנתונים הנדרשים לעיבוד באותו זמן. בנוסף קיים מספר רב של יחידות אחסון חיצוניות המשמשות לאיחסון ספריות התוכניות וכמויות גדולות של נתונים הנדרשים במערכת העיבוד. ככל שתדירות הפניה לנתונים גבוהה יותר נאחסן אותם ביחידות אחסון מהירות יותר.

קיימת מגמה ברורה של ירידה במחירי האחסון הפנימי ולכן אנו עדים להופעת מערכות מחשב בעלי כושר אחסון פנימי הולך וגדל (לדוגמא מספר מיליוני בתים). הדבר מסייע לניהול הפעלת המערכת, להפעלת מספר רב יותר של תוכניות ומשתמשים במקביל אך **לא** מיועד לאחסון קבוע של נתונים אשר נעשה ביחידות האחסון החיצוניות.



תרשים 2-3: הירידה בעלות אחסון בזכרון הפנימי

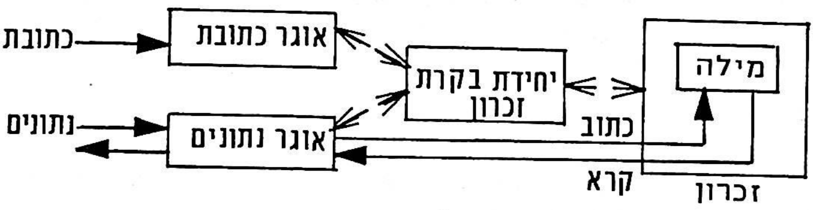
לסיכום נערוך השוואה בין אחסנה פנימית לאיחסון חיצוני:

תכונה	אחסנה פנימית	אחסון חיצוני
עלות	יקר ביותר	יקר פחות מאחסנה פנימית
קיבולת	מיליוני בתים	מאות מיליוני בתים ליחידה אחת
זמן גישה	מיליוניות שניה, ופחות	אלפיות שניה
עיבוד	מיידי	יש להעביר תחילה לאחסנה פנימית
שיטת אחסון	חצאי מוליכים, טרנזיסטורים	סרט מגנטי, דיסק מגנטי, תקליטון מגנטי, תוף מגנטי

2.2 אחסנה פנימית
(Internal Storage, Main Memory)

כל הנתונים המעובדים במחשב והתוכניות המבצעות את העיבוד באותה עת מאוחסנים באחסנה הפנימית של המחשב. יחידת האחסנה מורכבת ממספר רב של רכיבים אשר יכולים לאחסן ספרות בינאריות. כל יחידה נקראת תא זיכרון (memory cell). הטכנולוגיות המקובלות לבנית זכרונות היא שימוש במוליכים למחצה, טרנזיסטורים, בועות מגנטיות (ובעבר — טבעות מגנטיות) המאפשרים אחסון כמות גדולה של נתונים בנפח קטן. אחסנה פנימית של המחשב הינה מעשרות אלפי תאי אחסנה (בתים או מילים) עד מיליוני תאים במחשבים הגדולים.

מספר תאים בזכרון מרכיבים את היחידה הבסיסית **בית** (Byte) או **מילת המחשב** (Word). המילה יכולה לאחסן הוראה של המחשב או נתונים כלשהם. לכל בית או מילת מחשב יש **כתובת** שבעזרתה ניתן לפנות אליה באחסנה הפנימית.



תרשים 2-4 : פעולת האחסנה הפנימית

לקריאת תוכן מילת מחשב בזכרון, יש לשים את כתובתה באוגר הכתובת. יחידת בקרת הזכרון דואגת להעביר את תוכן המילה המבוקשת לאוגר הנתונים. לכתיבת מילה חדשה, יש לשים את כתובתה באוגר הכתובת ואת תוכנה באוגר הנתונים.

מהירות הגישה לכל מילה בזכרון נמדדת במיליוניות שניה (ואף פחות) ואינה תלויה במיקום המילה בזכרון. כפי שנראה בהמשך, תכונה זו אינה קיימת ביחידות אחסון אחרות.

ייצוג נתונים בזכרון

מחשבים פותחו בעיקר לטיפול במספרים, למרות שהיום רובם אינם עוסקים דוקא במספרים. הסיפרה הבינרית הינה המבנה הבסיסי והיסודי היחידי המוכר על־ידי המחשב. מכיון שהוא יכול לזהות רק שני מצבים — 0 או 1. לעבודה רגילה במחשב נדרשים כ־56 סימנים שונים שצריכים להיות מיוצגים במחשב: 10 ספרות, 26 אותיות וכ־20 סימנים מיוחדים. מכיון שאי אפשר לייצגם על ידי שני המצבים של 0 ו־1 דרושה שיטת ייצוג נרחבת יותר הקרויה BCD (Binary Coded Decimal). השיטה משתמשת בקבוצת סיביות לשם ייצוג כל אחד מהתוים. לייצוג 56 סימנים שונים דרושות 6 סיביות ($2^6 = 64$). רוב המחשבים משתמשים בקבוצות של 7 סיביות ($2^8 = 128$) כדי לאפשר ייצוג סימנים נוספים (כגון סימנים גראפיים או תווי בקרה מיוחדים למערכות תקשורת).

הסימנים המוזנים למחשב באמצעי הקלט השונים עוברים תרגום בעזרת טבלה לקוד BCD. במערכות יבמ למשל משתמשים בצופן EBCDIC בן 8 סיביות לייצוג הנתונים בזכרון. הצופן קובע את משמעות כל אחד מהצירופים השונים שניתן ליצור בעזרת 8 סיביות.

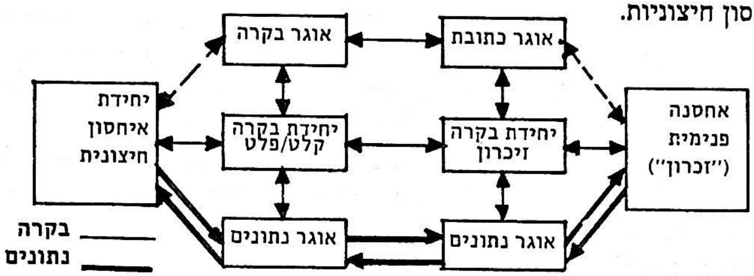
לדוגמא:

תו	צופן EBCDIC
2	11110010
F	11000110
+	01001110

קיימים מספר מבנים לאחסון נתונים מספריים. קיימות פקודות מיוחדות לטיפול בכל אחד מהמבנים ולביצוע הסבה ממבנה אחד למשנהו.

2.3 מבוא ליחידות אחסון חיצוניות

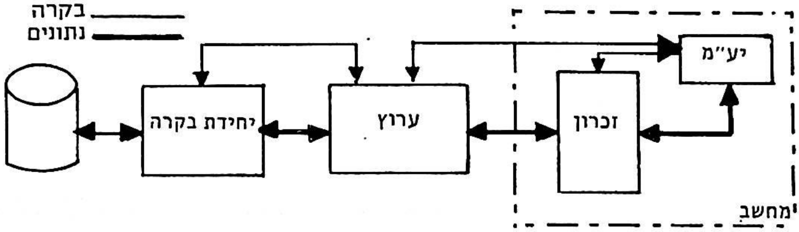
יחידות האחסון החיצוניות (Auxilary Storage) משמשות כהרחבה לכושר האחסון של זכרון המחשב. נתאר את שיטת העבודה של מחשב עם יחידות אחסון חיצוניות.



תרשים 2-5: פעולת יחידת אחסון חיצונית

באוגר הבקרה של יחידת האחסון החיצונית יש לשים את המידע על סוג הוראת קלט/פלט שיש לבצע ואת כתובת הנתונים. בזמן ביצוע ההוראה עוברים הנתונים אחד אחר השני לאוגר הנתונים, ומשם לזכרון או ליחידת אחסון. על כל הפעולות האלה מפקחת יחידת הבקרה המרכזית. בזמן ביצוע פעולת קלט/פלט ביחידות האחסון החיצוניות, היע"מ נמצא במצב של המתנה לגמר פעולת הקלט/פלט. מהירות העיבוד ביחידות האחסון, שהן מכניות בעקרון פעולתן, איטית בהרבה בהשוואה למהירות העיבוד של היע"מ ולכן יש בשיטת עבודה זו אלמנט רציני של בזבוז משאבים. בהמשך נעמוד על השיטות ליעול עבודת המחשב בתהליך קלט/פלט.

<u>**ערוץ קלט/פלט** (Input-Output Channel):</u> במחשבים זעירים וקטנים מטפל היע"מ בפעולות עיבוד וגם מוקדשים למטרת קלט/פלט יחידות מיוחדות או אף מחשב מיוחד הנספח למחשב המרכזי. המחשב המרכזי מוקדש בעיקר לביצוע פעולות עיבוד הנתונים והמחשב הקטן (הנקרא גם ערוץ קלט/פלט), מטפל בכל נושא הקלט/פלט. במחשבים בינוניים וגדולים מותקנים מספר ערוצים או מחשבי קלט/פלט בכדי ליעל את תהליך זרימת הנתונים בין האחסנה הפנימית לבין יחידות האחסון החיצוניות המשרתות תוכניות רבות בעת ובעונה אחת.



תרשים 2-6: פעולת מחשב עם ערוץ קלט/פלט

פעילות קלט/פלט יזומות על־ידי היע"מ בעזרת פקודות מיוחדות. הערוץ מתקדם כמו מחשב רגיל ופונה אל הזכרון המיכני, בו נמצאת תוכנית מחשב מיוחדת (Channel Program) לביצוע תהליך קלט/פלט המבוקש על ידי תוכנית המשתמש. תוכנית הערוץ קובעת לערוץ איזה פעולת קלט/פלט יש לבצע והכתובת בזכרון ממנה יש לקחת את הנתונים או אליה יש להעביר אותם. גם הערוץ וגם היע"מ פונים אל אותו זיכרון (Memory Sharing). לקבלת ההוראות ולקריאת וכתובת הנתונים בהתאם להוראות אלו. תוכנית הערוץ מתבצעת במקביל לעבודת היע"מ ולפיכך הם "מפריעים" זה לזה בעת גישה לזכרון. הם "חוטפים" זה מזה פרקי זמן (Cycle Stealing) לשם ביצוע העבודה. הערוץ משתמש בפקודות **התערבות** מיוחדות (Interrupt) כדי להודיע ליע"מ על אירועים מיוחדים.

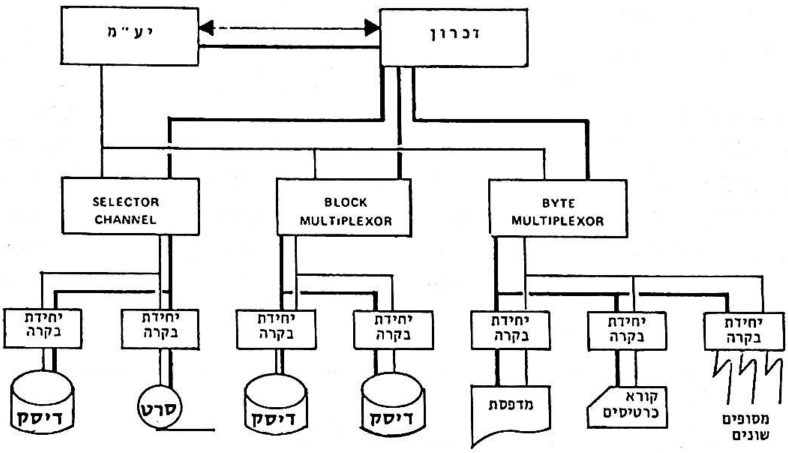
לכל יחידת אחסון יש יחידת בקרה מיוחדת (Device Controller) המאפשרת העברת אינפורמציה בקרה אל יחידת האחסון וממנה, קובעת עדיפויות ומספקת לערוץ **קודי מצב** (Status Signals). כאשר יחידת הבקרה מודיעה לערוץ שפעולת קלט/פלט הסתימה, מבצע הערוץ פעולת התערבות ומודיע ליע"מ שהפעולה המבוקשת הסתימה. הערוץ יכול להכיל **מאגר** (Buffer) כמישק בין הזכרון לבין יחידת האחסון. המאגר מתאם בין סוגי האינפורמציה המטופלים בזכרון וביחידת האחסון (למשל בזכרון מטפלים במילת מחשב המכילה מספר בתים בעוד שיחידת האחסון מטפלת רק בבית בודד).

קיימים מספר סוגי ערוצים, המסווגים בהתאם למהירויות העבודה שלהם. במחשבי יבמ מבחינים בין שלשה סוגי ערוצים:

— ערוץ SELECTOR — המיועד לטיפול ביחידות אחסון מהירות כגון דיסקים וסרטים מגנטיים. ערוץ זה יכול לטפל בו זמנית רק בפעולת העברה אחת.

— ערוץ BYTE MULTIPLEXOR — המיועד לטיפול במספר יחידות קלט/פלט איטיות כגון קורא כרטיסים, מדפסת, מסופים.

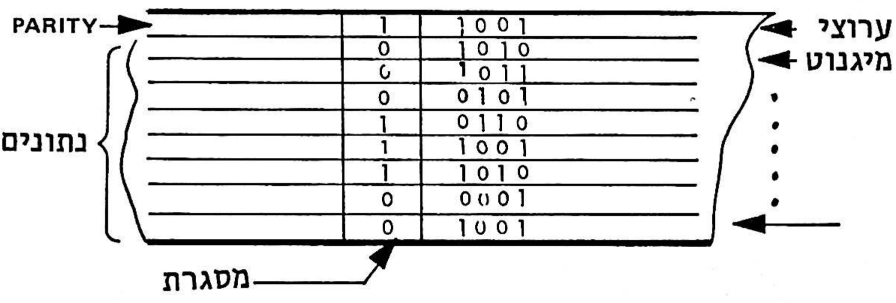
— ערוץ BLOCK MULTIPLEXOR — המשלב את מהירות העבודה של ערוץ SELECTOR עם אפשרות הטיפול במקביל במספר יחידות אחסון.



תרשים 2-7: אירגון ערוצי קלט/פלט במחשב יבמ

2.4 סרט מגנטי (Magnetic Tape)

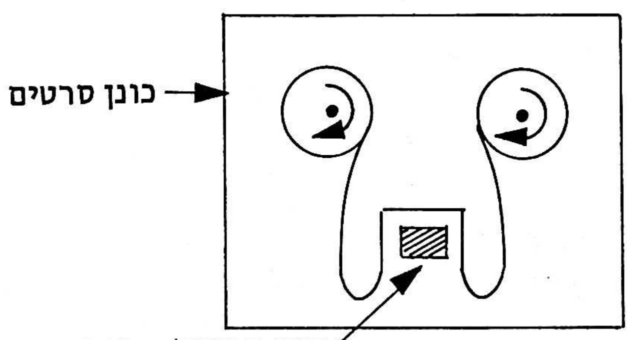
הסרט המגנטי מורכב מסרט פלסטי המצופה בחומר הקולט מיגנוט. הסרט הפלסטי כרוך על גבי גלילים בקוטר של כ־10 אינץ'. לרוחב הסרט יש **מסגרות** (Frames) המכילות מספר קבוע של סיביות. הנפוצים ביותר הם 7 ו־8 סיביות למסגרת. בכל מסגרת ניתן למגנט אות אחת או שתי ספרות. למספר המסגרות לאינץ נקרא **צפיפות המיגנוט** (BPI — Bytes Per Inch). צפיפות המגנוט המקובלת היא 800, 1600 ו־6250 מסגרות לאינץ אחד. כלומר, באינץ אחד של אורך סרט אפשר לאחסן 800, 1600 או 6250 תוים, בהתאמה.



תרשים 2-8 : מבנה הסרט המגנטי

במסגרת אחת נמצא סיביות נתונים (8, כאשר יחידת המידע היא "בית") וסיבית אחת נוספת לצורך ביקורת תקינות (ביקורת זוגיות — Parity Bit).

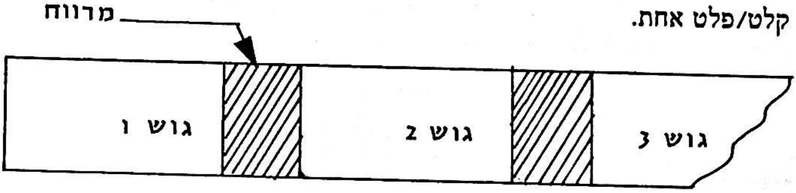
הסרט המגנטי מורכב על גבי **כונן סרטים** (Tape Drive). הוא מתקדם לכיוון אחד בלבד ואין אפשרות לקרוא את הנתונים בכיוון השני. הסרט חייב לנוע במהירות קבועה מתחת לראש הקריאה/כתיבה.



תרשים 2-9 : מבנה סכימטי של כונן סרט מגנטי

ראש הקריאה/כתיבה סורק את המסגרות ומפענח את הצירופים הבינריים המופיעים בכל מסגרת ומעביר אותן אל זכרון המחשב לעיבוד. מאחר והסרט יכול להכיל מספר רב מאד של מסגרות, כמות הנתונים שניתן לאחסן בו היא גדולה ונמדדת בעשרות מיליוני תוים. מאחר ואין אפשרות להכניס את כל הנתונים שבסרט לזכרון המחשב, יש לחלק את הנתונים בסרט ליחידות קטנות יותר הנקראות **גושים** (Blocks) ולקרוא אותם לזיכרון ביחידות אלו. גם הכתיבה נעשית ביחידות של גושי נתונים ולא ברצף אחד.

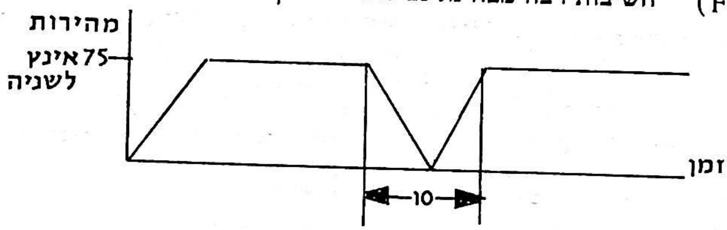
הגדרה: גוש הינו אוסף של רשומות המועברות אל הזיכרון וממנו בפעולת קלט/פלט אחת.



תרשים 2-10: חלוקת נתונים בסרט מגנטי לגושים

הנתונים שנקראים לזיכרון עוברים תהליך עיבוד הנמשך זמן כלשהו. בפרק זמן זה יש לאפשר לכונן הסרטים לעצור ולהמתין לגמר העיבוד. מאחר והסרט חייב לנוע במהירות קבועה כדי שניתן יהיה לקרוא/לכתוב, הרי שיש להאיץ אותו למהירות הקריאה ולהאיט לקראת עצירה. בפרק זמן זה אי אפשר לקרוא או לכתוב בסרט. פרק זמן זה נמשך כ־10 מילישניות בסרט הנע במהירות של 75 אינץ לשניה. כדי להתגבר על בעיה זו, יש ליצור **מרווח** (Gap) על הסרט שאינו כולל נתונים ותפקידו להפריד בין גוש לגוש.

בתוך הגוש נמצאות הרשומות. **למספר הרשומות בגוש** (Blocking Factor) חשיבות רבה מבחינת נצילות האחסון בסרט ויעילות העיבוד.



תרשים 2-11: מהירות תנועת הסרט כפונקציה של הזמן

נסמן ב-B את גודל הגוש וב-R את גודל הרשומה:

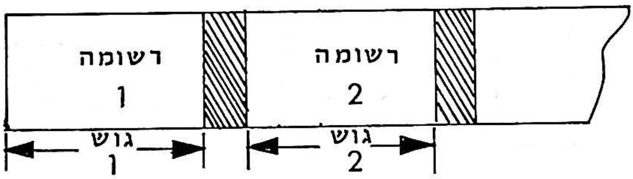
$$\frac{B}{R} = \frac{\text{גודל גוש}}{\text{גודל רשומה}} = \text{מספר רשומות לגוש}$$

לדוגמא: אם גודל רשומה הוא 100 בתים ומספר הרשומות לגוש הוא 10, נקבל גודל גוש של 1000 בתים.

ככל שנבנה גושים קטנים יותר, נקבל מספר רב יותר של מרווחים ולכן ניצול גרוע יותר של שטח האחסון. מאידך ככל שגודל הגוש גדול יותר, נצילות שטח האיחסון גבוהה יותר אבל יהיה צורך במאגרים גדולים בזכרון לקליטת כל הגוש. ראה בנספח א׳ שיטה לחישוב גודל הגוש.

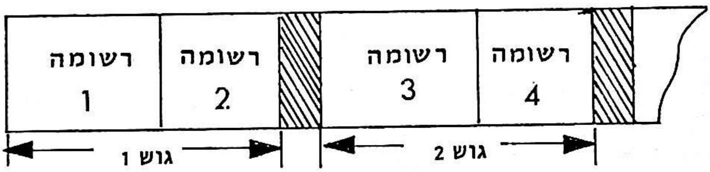
קיימות מספר תבניות קבועות לרישום הגושים בסרט מגנטי. נציג כאן את העיקריות:

1. Unblocked Fixed — רשומה לגוש. הרשומות באורך קבוע.



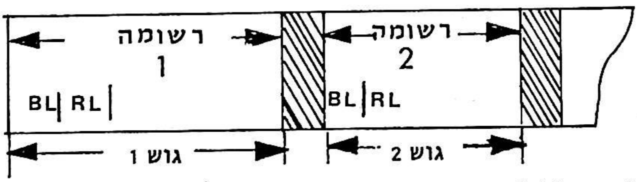
תרשים 2-12: תבנית רשומה באורך קבוע לגוש

2. Blocked Fixed — מספר רשומות באורך קבוע לגוש.



תרשים 2-13: מספר רשומות באורך קבוע לגוש

3. Unblocked Variable — רשומה אחת לגוש. הרשומה באורך משתנה. מוסיפים לגוש הנתונים שדה לציון גודל הגוש (Block Lenght — BL) וגודל הרשומה (RL — Record Lenght).

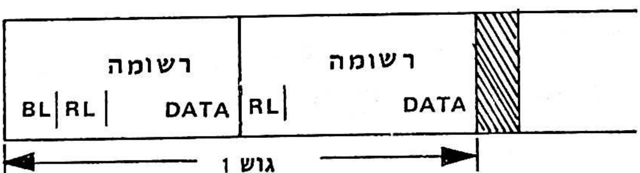


BL = RL

תרשים 2-14: תבנית עם רשומה משתנה לגוש.

4. Blocked Variable — כל גוש מכיל מספר רשומות באורך משתנה.

בנוסף לתבניות אלה קיימות תבניות נוספות: Spanned, Undefined.



$$BL = \sum_{i=1}^{n} RL_i$$

תרשים 2-15: תבנית עם מספר רשומות משתנות לגוש

מהירות העברת נתונים

אחד המאפיינים החשובים ביותר של סרטים מגנטיים הוא מהירות העברת הנתונים בין הכונן לזכרון המחשב. מהירויות אלה יכולות לנוע בין עשרות אלפי תווים לשניה ועד ליותר ממיליון תווים לשניה, בהתאם לציוד. (BPS — Bytes Per Second). נסמן ב־t את מהירות ההעברה של כונן סרטים. זמן קריאת גוש יהיה:

$$T_{B1} = \frac{B}{t} = \frac{\text{גודל גוש}}{\text{מהירות העברה}}$$

מאחר ובין גוש לגוש קיים זמן עצירה, יש להוסיף לכל גוש את זמן העצירה וההתנעה (START/STOP Time). סה"כ זמן קריאת הגוש יהיה:

$$T_B = \text{S/S time} + \frac{B}{t}$$

דוגמא:

	סוג כונן	מספר ערוצי מגנוט	צפיפות מיגנוט ב־BPI	מהירות הסרט באינץ לשניה	מהירות העברה ב־BPS (סיביות לשניה)
יבמ 3420 ←	מודל 3	9	1600	75	120,000
יבמ 3420 ←	מודל 7	9	1600	200	320,000
יבמ 3420 ←	מודל 8	9	6250	200	1,250,000

חישוב קיבולת אחסון

אחד הנתונים החשובים בסרט מגנטי הוא כמות הנתונים נטו (כלומר, בניכוי המרווחים שאינם ממוגנטים) שניתן לאחסן בו. מאחר וגודל המרווח בין הגושים הוא קבוע (בדרך כלל 0.6 אינץ) ניתן לחשב את מספר הגושים בסרט על פי הנוסחה הבאה:

$$\text{מספר גושים} = \frac{\text{אורך הסרט באינץ}}{\text{אורך הגוש באינץ} + \text{גודל המרווח באינץ}}$$

הערה: מאחר ובדרך כלל אורך הסרט נתון ב"רגל" (foot), יש לכפול אותו ב־12 לקבלת אורכו באינץ (1 foot = 12 inch).

את אורך הגוש ניתן לחשב על סמך מספר התווים לגוש ולפי צפיפות המיגנוט.

$$\text{אורך גוש באינץ} = \frac{\text{מספר תווים בגוש}}{\text{צפיפות מיגנוט}}$$

נציב בנוסחה קודמת ונקבל את הנוסחה לחישוב מספר הגושים בסרט:

$$\text{מספר גושים} = \frac{\text{אורך סרט באינץ}}{\dfrac{\text{מספר תווים לגוש}}{\text{צפיפות מיגנוט}} + \text{גודל המרווח באינץ}}$$

כמות הנתונים בסרט תהיה מכפלת מספר הגושים במספר התווים לגוש. ככל שהגוש גדול יותר עולה הכמות המנוצלת של הסרט. החל מגודל מסוים, הנצילות עולה באופן איטי ולכן לא כדאי להשתמש בגושים גדולים מאד. אמנם הנצילות תעלה, אבל שטח המאגרים יגדל בהרבה ותתעוררנה בעיות אחרות הקשורות בחלוקת הזכרון והקצאתו לשטחי תוכנית ומאגרים ויעילות ניצול הערוצים.

חישוב גודל הגוש בסרט מגנטי

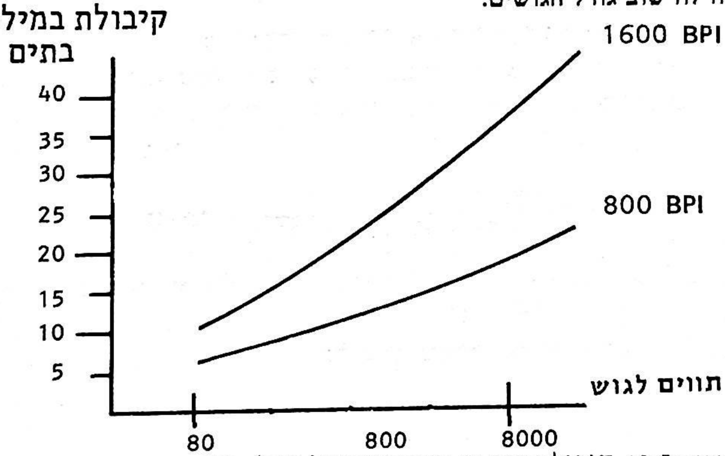
אחד הפרמטרים החשובים בשימוש בסרטים מגנטים הוא גודל הגוש. גודל הגוש משפיע על:

— **מספר הגושים בסרט** ולכן על מספר פעולות הקלט/פלט הדרושות לקריאת הקובץ. ככל שהגושים גדולים יותר, מספר הגושים בקובץ קטן ולכן דרושות פחות פעולות קלט/פלט.

— **ניצול שטח האחסון בסרט.** ככל שהגושים גדולים יותר, ניצול הסרט מבחינת אחסון, טוב יותר מאחר ומספר המרווחים קטן יותר.

— **שטח המאגרים הדרושים בזכרון.** ככל שהגוש גדול יותר, שטח המאגר צריך להיות גדול יותר, ולכן נדרש שטח גדול יותר עבור התוכנית.

קיימות מערכות לניהול קבצים המכתיבות את גודל הגוש למשתמש. גודל זה מבוסס על שיקולים של יעילות מבחינת הציוד וניצולו. במקרים אלה יש לנסות לבנות רשומות וגושים הקרובים ככל הניתן לגושים אלה. אם לא קיימת מגבלה מסוג זה, על המשתמש לבנות את הגושים על סמך שיקולים שלו. בנספח א' מוצג דיון מעמיק יותר בנושא יעילות העיבוד כפונקציה של גודל הגוש וניתנת שיטה לחישוב גודל הגושים.

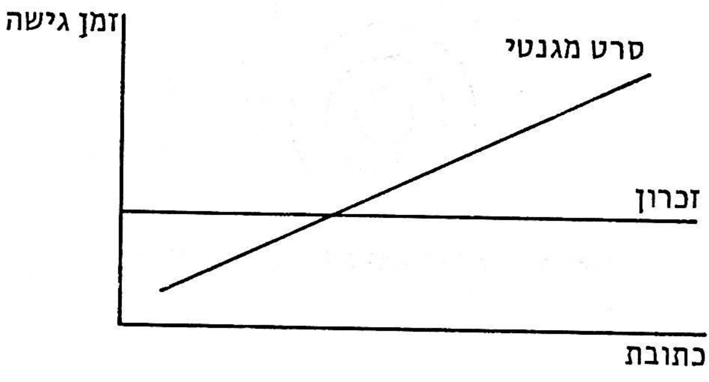


תרשים 2-5: קיבולת אחסון כפונקציה של גודל הגוש וצפיפות המיגנוט (2400 רגל, מרווח 0.6 אינץ)

סיכום

הסרט המגנטי הינו אמצעי אחסון זול ונפוץ ביותר. שימושו העיקרי הוא במערכות העובדות באצווה (Batch) ולגיבוי קבצים הנמצאים בדיסקים מגנטיים. מהירות העברת הנתונים גבוהה ויכולה להגיע למיליון תוים לשניה, ויותר. קיבולת אחסון יכולה להגיע לכמה עשרות מיליוני תווים. אחסון הסרטים בחדר המחשב נוח בגלל נפחם הקטן יחסית. אמינותו מוגבלת בגלל השימוש בסרט פלסטי היכול להקרע או לקבל עיוות מפאת מתיחה פתאומית או בלאי שימוש. הסרט המגנטי מאפשר קריאה או כתיבה אבל לא את שניהם ביחד, בגלל אפשרות תנועת הסרט בכיוון אחד בלבד. כלומר, ניתן לקרוא או לכתוב אך לא לכתוב ולקרוא לסירוגין ואין אפשרות לקרוא ואח"כ לכתוב שוב באותו גוש.

החסרון הגדול ביותר של סרטים מגנטים הוא שאין אפשרות לגישה ישירה לרשומות. זמן הגישה לרשומה נמצא באופן יחסי למיקום הרשומה בסרט מכיון שהחיפוש של הרשומה הוא סדרתי. זמן הקריאה של סרט המכיל 30 מיליון תווים יכול להגיע ל־4 דקות, לכן אין אפשרות להשתמש בסרט כאשר יש צורך בזמני גישה קטנים.



תרשים 2-16: זמן הגישה כפונקציה של כתובת הרשומה

הביצועים של הסרט המגנטי תלויים בפרמטרים הבאים:

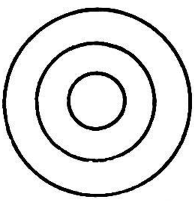
- צפיפות מגנוט,
- מהירות תנועה, } מגדירים את מהירות ההעברה
- גודל מרווח,
- זמן האטה/תאוצה,
- מהירות גלגול לאחור,
- אפשרות לקרוא או לכתוב בשני הכיוונים.

2.5 דיסק מגנטי (Magnetic Disk)

הדיסק המגנטי הינו יחידת אחסון בעלת תכונות פיסיות המאפשרות גישה ישירה לכל יחידת אינפורמציה בהתאם לכתובת. בגלל תכונה זו נהוג לקרוא להם **יחידות אחסון בגישה ישירה** (Direct Access Storage Device — DASD) מאחר ואין צורך בביצוע סריקות לגישה לנתונים בדומה לסרט.

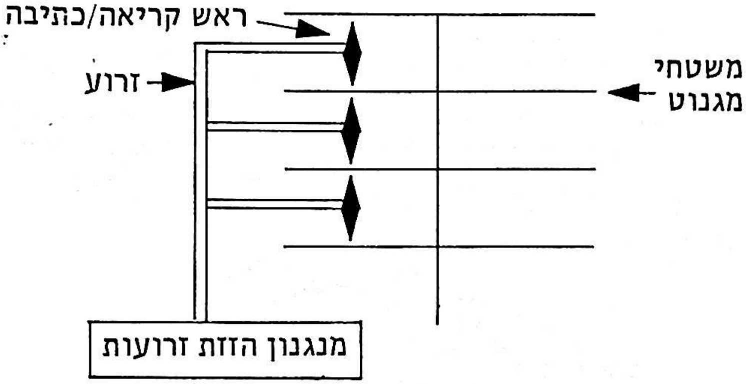
מבחינת עקרון הפעולה, דומה הדיסק לתקליטי המוסיקה המנוגנים בבתינו. הנתונים נקראים או נכתבים תוך כדי סיבובי הדיסק. יחידת האחסון מורכבת משלשה מרכיבים עיקריים:

— **משטחי מגנוט:** משטחי המגנוט נמצאים משני צידי הדיסק ומשמשים לרישום הנתונים. יחידת דיסקים אחת (Disk Pack) מכילה מספר דיסקים השונה מיחידה ליחידה. לדוגמא, יחידת דיסקים 3330 של חברת יבמ מכילה 10 דיסקים ויחידה 3340 מכילה 6 דיסקים. שטח התקליט מחולק **למסילות** (Tracks) קונצנטריות. כל מסילה מכילה את אותה כמות הנתונים ללא תלות בקוטר שלה. לדוגמא דגם 3330 של חברת יבמ מכיל מעל 400 מסילות בדיסק.

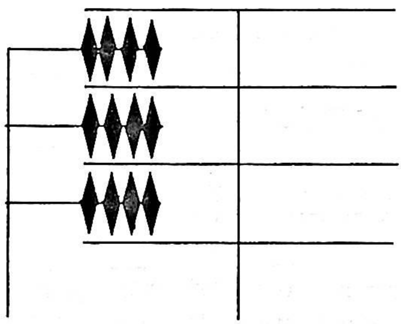


תרשים 2-17: חלוקת משטח המגנוט למסילות

— **ראשי קריאה/כתיבה:** לכל משטח מגנוט יש ראש קריאה/כתיבה המאפשר קריאה וכתיבה של הנתונים. כל ראשי הקריאה/כתיבה של כל משטחי המגנוט מחוברים למנגנון הזזה אחד ולכן נעים כולם יחד. תנועת הזזת הזרועות היא תנועה איטית יחסית (עשרות אלפיות שניה). כדי להתגבר על בעיה זו קיימות יחידות אחסון המשתמשות בראש קריאה/כתיבה קבוע לכל מסילה ולכן אין צורך בביצוע ההזזות (לדוגמא, דגם 3380 של יבמ מכיל גם ראשי קריאה/כתיבה קבועים). בשיטה זו זמן הגישה לנתונים מהיר ביותר. החסרון של ראשים קבועים שלא ניתן לשלוף את הדיסק ולהחליפו באחר ולכן מחירו גבוה יחסית לדיסק עם מנגנון להזזת זרועות.



תרשים 2-18: יחידת דיסקים עם מנגנון להזזת זרועות



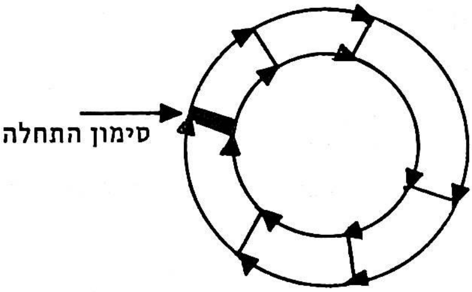
תרשים 2-19: דיסק עם ראשים קבועים

— **כונן הדיסק** (Disk Drive): הכונן מכיל את תושבת יחידת הדיסקים, את המנגנון המסובב אותה במהירות סיבוב קבועה ואת מנגנון ראשי הקריאה/כתיבה. אם הזרועות נעות, ניתן לשלוף את יחידת הדיסקים ולהחליפה ביחידה אחרת.

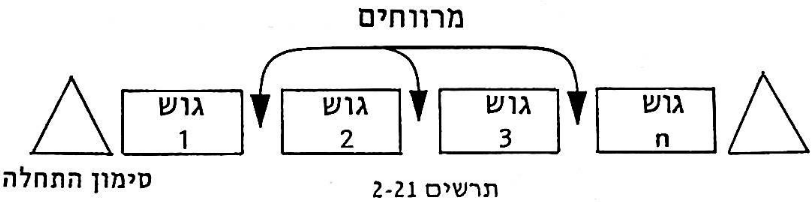
היתרון של הדיסקים נובע מאפשרות הגישה הישירה והמהירה לנתונים ללא קשר למיקומם. יתרון נוסף הוא האפשרות להשתמש בהם לקלט ופלט לסירוגין בתהליך עיבוד אחד ולכתוב גושים לאחר קריאתם ועדכונם. (בסרט מגנטי ניתן להשתמש לכתיבה או לקריאה אך לא לסירוגין). במערכות תקשורות מקוונות חייבים להשתמש בדיסק בגלל דרישות זמן התגובה המהיר שלהן.

שיטת רישום הנתונים במסילה

הנתונים נרשמים במשטח הדיסק במסילות קונצנטריות (מסילות מעגליות עם מרכז אחד אך קטרים שונים). לכל מסילה יש נקודת התחלה המסומנת עליה (Index Marker). המסילה מכילה גושים שמספרם וגודלם נקבע על־ידי המתכנת או נקבע מראש על־ידי היצרן בהתאם לסוג הדיסק.

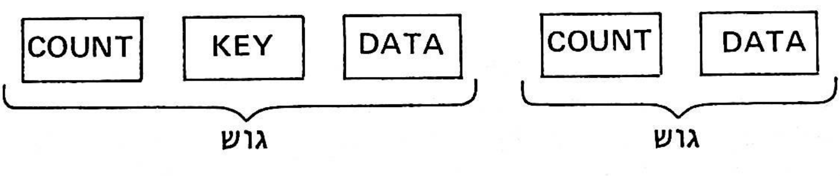


תרשים 2-20: מבנה מסילה וסימון התחלה



תרשים 2-21

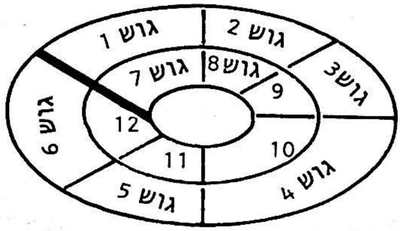
בכל מסילה מופיע סימון התחלת המסילה ואחריה גוש מיוחד המתאר את המסילה (כתובתה ומצבה, Track Descriptor) ומשמש את מערכת ההפעלה ואת התוכנה לניהול הנתונים. בין הגושים קיימים מרווחים המשמשים לביצוע פונקציות בקרה של התוכנה. קיימים שני סוגי גושים — גוש המכיל מפתח וגוש ללא מפתח.



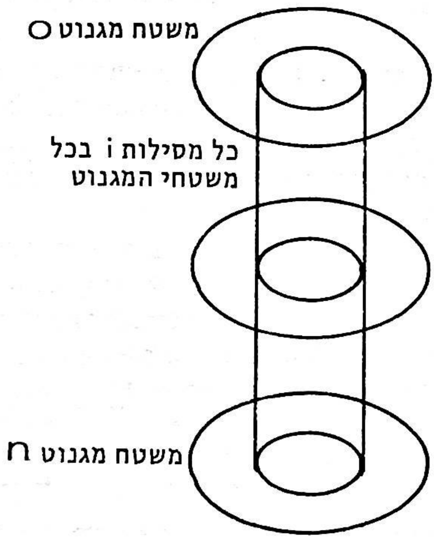
תרשים 2-22: תבנית הגוש

בכל גוש ישנו **שדה מונה** (Count) המשמש לאיתור הגוש. אם משתמשים בשיטות אירגון המאפשרות גישה לפי מפתח ניתן לשמור את **המפתח** (Key) בקטע מיוחד בגוש ליעול החיפוש אחר רשומות. כתובת רשומה בדיסק נקבעת על־ידי מספר משטח המגנוט, מספר מסילה, המספר הסודר של הגוש במסילה וזיהוי הרשומה או מפתח שלה.

את הנתונים בדיסק ניתן לרשום באופן סידרתי, גוש אחר גוש, מסילה אחר מסילה עד לגמר משטח מגנוט ראשון ולהמשיך למשטח מגנוט שני. שיטה זו מאד לא יעילה מאחר והיא מחייבת ביצוע הזזות זרוע רבות לקריאת הקובץ. ניתן לרשום את הנתונים בצורה שונה: גוש אחר גוש עד גמר המסילה, לעבור למסילה זהה במשטח מגנוט שני וכך הלאה. בשיטה זו ניתן לקרוא את הקובץ ללא הזזות זרוע רבות כפי שהיה במקרה הקודם. אוסף המסילות הנקראות/נכתבות ללא הזזת זרוע מהווה גליל (Cylinder).



תרשים 2-23: רישום נתונים רציף במסילות



תרשים 2-24: מבנה הגליל ביחידת דיסקים

אם הקובץ מכיל מספר רב של נתונים הם ירשמו במספר גלילים. במקרה זה יהיה צורך בביצוע הזזות זרוע לקריאת הקובץ. בדיסקים של יבמ מקובלים הפרמטרים הבאים:

	קיבולת (במיליוני בתים)	זמן גישה ממוצע (במילי שניות)	מהירות העברה (באלפי בתים לשניה)
כונן 3310	64.5	27	1031
כונן 3340	70 או 35	25	885
כונן 3350	635	25	1198
כונן 3370	571	20	3000

פרמטרים אופיניים לדיסקים מגנטיים:

	יחידת דיסקים קטנה	**יחידת דיסקים גדולה**
זמן גישה במילישניות	30	60
מספר גלילים לתקליט	200	400
מהירות העברה בתווים לשניה	312.000	806.000
קיבולת במיליוני תווים	29-58	100-200

סוגי הגושים בדיסק

הגוש הינו אוסף של רשומות לוגיות ומטופל על־ידי מערכת ההפעלה כיחידה אחת. ניתן לעבוד עם גושים המכילים **רשומה לוגית אחת** (Unblocked) או גושים המכילים **מספר רשומות** (Blocked). השימוש המקובל ביותר הוא עם מספר רשומות לגוש. השיטה מאפשרת:

— קריאה/כתיבה מהירה יותר מאחר ויש פחות פעולות קלט/פלט,
— ניצול טוב יותר של יחידת האחסון מאחר ויש פחות מרווחים בין גושים,
— ניתן ליעל תהליכי חיפוש על־ידי איתור הרשומה האחרונה בגוש.

השימוש ברשומה אחת לגוש פחות נפוץ. משתמשים בו בעיקר עבור רשומות גדולות או אם אין מידע על אורך הרשומה.

להלן מספר תבניות לרישום נתונים בדיסק:

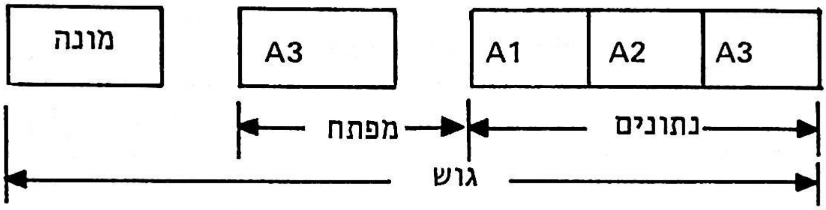
.1 Fixed Unblocked — רשומה באורך קבוע לגוש.



תרשים 2-25

חלק המפתח וחלק הרשומה הם בעלי אורך קבוע, ואם המפתח קטן יותר תבוצע השלמה במרווחים. קיימת אפשרות לא לחזור על המפתח בתוך הרשומה. החיפוש מבוצע ע״י מערכת ההפעלה על פי המפתח. אם שיטת הארגון אינה כוללת מפתח אין צורך להשתמש בתת־גוש המכיל את המפתח.

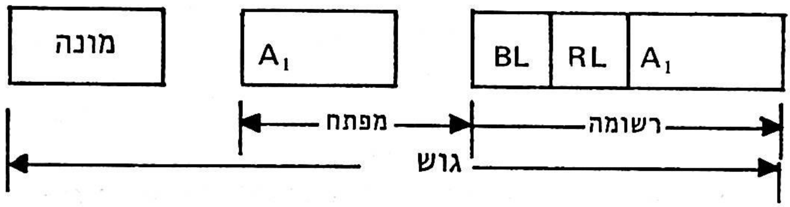
.2 Blocked Fixed: מספר רשומות באורך קבוע לגוש.



תרשים 2-26

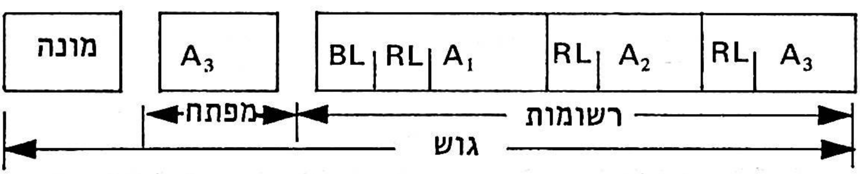
היתרון של שיטה זו הוא שיש מונה אחד עבור קבוצה של רשומות לוגיות ולכן יש פחות מרווחים. בתת־גוש המפתח ניתן לשמור את המפתח של הרשומה הגבוהה ביותר בגוש ובצורה זו לפשט את תהליכי החיפוש.

.3 Unblocked Variable: רשומה בעלת אורך משתנה לגוש.



תרשים 2-27

.4 Blocked Variable: מספר רשומות בעלות אורך משתנה לגוש.



תרשים 2-28

זמן גישה בדיסק

זמן הגישה בדיסק מורכב משלוש יחידות זמן:

— **זמן חיפוש** (Seek Time): הזמן הדרוש להזזת ראשי הקריאה/כתיבה למסילה המבוקשת. ככל שמספר המסילות שיש לחלוף על פניהן גדול יותר, גם זמן החיפוש יהיה גדול יותר.

— **זמן השהייה סיבובית** (Rotational Delay): הזמן העובר מהרגע בו הגיע ראש הקריאה/כתיבה למסילה המבוקשת ועד שהוא נמצא מעל הגוש המבוקש. בממוצע זמן זה שווה למחצית הזמן הדרוש להשלמת סיבוב מלא של הדיסק. זמני השהייה מקובלים הם בין 8 ל־10 מילישניות.

— **זמן העברת נתונים** (Transfer Time): הזמן הדרוש להעברת הגוש מכונן הדיסק לזכרון המחשב, ולהיפך. זמן זה תלוי בסוג הכונן בו משתמשים. מהירויות קריאה/כתיבה מקובלות הן בין 100 אלף בתים לשניה ועד כ־3 מיליון בתים לשניה. כפי שניתן לראות, מהירויות אלה גבוהות בדרך כלל ממהירויות ההעברה של סרטים מגנטים. מהירות ההעברה אינה תלויה במיקום המסילה על פני משטח המגנוט.

במציאות משתמשים בנתוני היצרנים לחישוב זמן גישה ממוצע או בשיטות חישוב לא מדויקות להערכתם. זמן הגישה לנתונים יכול לנוע בין אלפיות שניה ועד עשיריות ויותר. כאשר נדרשות גישות אחדות (ראה את הסיבה לכך בהמשך) יכול הזמן הכולל להגיע לשניה אחת ויותר. קיימים מקרים בהם אומדן לא נכון של זמן הגישה יכול להוביל לתכנון לא נכון של מערכות תקשורת ולהתמוטטותן בגלל זמני תגובה בלתי מספיקים. ראה דיון נוסף על זמני גישה בנספח ב'.

סיכום

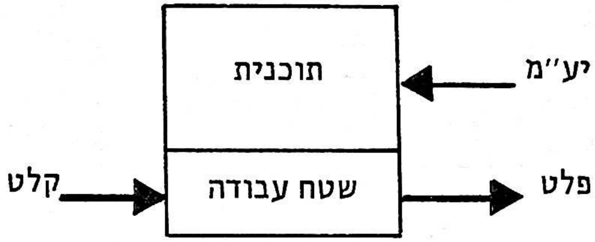
הדיסק המגנטי הינו אמצעי אחסון נפוץ מאד ושימושי בעיקר במערכות מקוונות בהן יש צורך בזמני גישה קצרים מאד. מהירות העברת הנתונים של הדיסקים גבוהה יחסית לסרטים מגנטים ויכולה להגיע לכמה מיליוני תווים לשניה. קיבולת האחסון נעה בין מספר עשרות מיליוני תווים ועד מיליארד תווים ביחידות החדשות. אמינות הדיסק המגנטי כאמצעי לאחסון נתונים גבוהה מאד יחסית לסרטים מגנטים מאחר והבלאי שלהם קטן מאד. הדיסק מאפשר גם קריאה וגם כתיבה באותו קובץ.

היתרון הגדול של הדיסקים הוא זמני הגישה המהירים לרשומה. זמן ממוצע לגישה לרשומה הוא 25 מילישניות. חסרונו הגדול של הדיסק הוא מחירו הגבוה לאחסון יחידת מידע. המחיר הגבוה הביא לנסיונות פיתוח של יחידות הדומות בעקרון פעולתם לדיסק אבל זולות ממנו. התפתחות הטכנולוגיה מציעה מדי פעם דיסקים בעלי תכולה גדולה יותר במחיר נוח יותר ועל ידי כך מאפשרת ניצול היתרונות הרבים בתחומים מגוונים יותר ולפיתוח יישומים חדישים המחייבים מערכות אחסון בגישה ישירה.

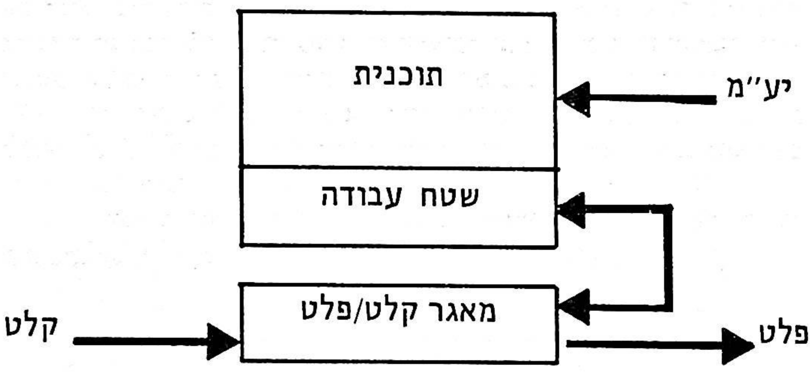
במחייבים קטנים וזעירים מקובל השימוש **בתקליטונים** (Floppy Disks) למטרת אחסון נתונים בגישה ישירה. עקרון פעולתם דומה לפעולת הדיסקים שהזכרנו קודם אך מהירותם וקיבולם שונים באופן משמעותי. התקליטון מהווה פשרה בין הדיסק המגנטי שמחירו אלפי דולרים לבין הסרט המגנטי שמחירו נמוך. התקליטון מסוגל לאחסן בין 250,000 ל־500,000 בתים. זמן גישה ממוצע הוא פחות משניה, מהירויות העברה מקובלות בין 25,000 ל־50,000 בתים לשניה.

2.6 מאגרי קלט/פלט (I/O Buffers)

לעיבוד רשומה במחשב יש תחילה לקרוא אותה לתוך הזיכרון. בגמר העיבוד יש לכתוב אותה אל יחידת אחסון חיצונית. קיים הבדל ניכר בין מהירויות העבודה של היע"מ לבין מנגנוני הגישה ליחידות האחסון. בגלל הבדל זה על היע"מ לחכות לרשומה הבאה אם היא אינה נמצאת בזיכרון. במקום לקרוא רשומה בודדת לזכרון, ניתן לקרוא קבוצה של רשומות (גוש) לזיכרון, לשטח מיוחד הנקרא **מאגר** (Buffer), ו"להכין" אותן עבור היע"מ.

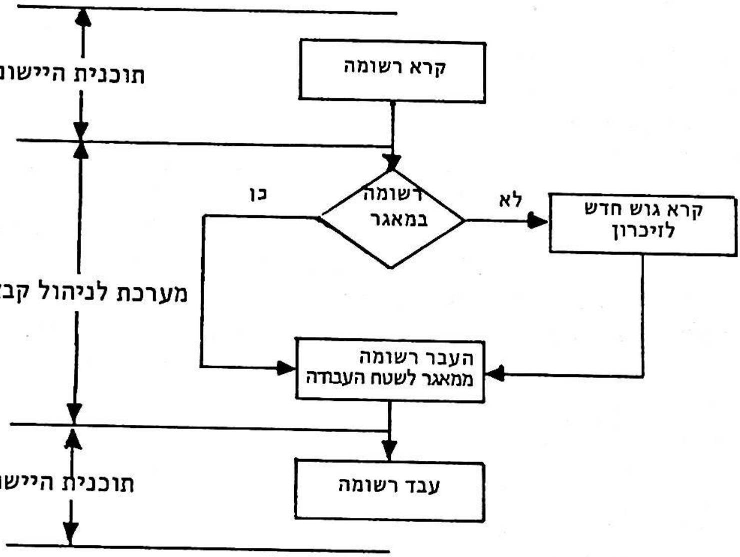


תרשים 2-29: עיבוד רשומה במחשב



תרשים 2-30: שימוש במאגר קלט/פלט

בשיטה זו, אין צורך בביצוע גישות ליחידת האחסון מיד עם גמר עיבוד הרשומה. ברגע שהתוכנית מבקשת את הרשומה הבאה, יש לבדוק תחילה אם הרשומה המבוקשת נמצאת במאגר. אם כן, כל מה שיש לעשות הוא להעביר אותה מתוך המאגר לשטח העבודה. אם הרשומה איננה במאגר, יש לבצע גישה ליחידת אחסון ולקרוא את הגוש הבא. שיטת ניהול הקלט דואגת לכך שתמיד יהיה בזיכרון גוש אחד נוסף, מראש, כדי למנוע עיכוב בעבודת היע"מ כאשר תדרש הרשומה העוקבת לעיבוד (תהליך דומה קיים בפלט, כפי שנראה בהמשך). לכל יחידת אחסון יש להקצות בזיכרון את המאגר שלה. ניהול המאגרים נמצא באחריות המערכת לניהול קבצים.

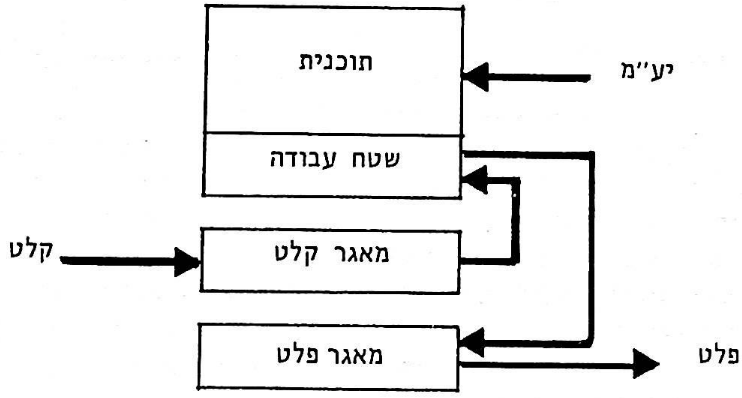


תרשים 2-31: שיטת העברת רשומות ממאגר לשטח עבודה

התהליך מורכב יותר כאשר לאחר הקריאה מעדכנים את הרשומות וכותבים את הגוש שוב בדיסק. ניהול מאגר אחד לכל יחידת אחסון אינה מאפשרת הבאת הגוש הבא לזכרון כל עוד הגוש המעובד לא נכתב ביחידת האחסון. ניתן להתגבר על בעיה זו על ידי הקצאת שני מאגרים בזכרון לכל יחידת אחסון:

— מאגר קלט לתוכו נקרא את הגוש החדש וממנו נעביר את הרשומות לשטח העבודה בתוכנית. עם גמר עיבוד כל הרשומות יש להביא גוש חדש לתוך מאגר הקלט.

— מאגר הפלט אליו נעביר רשומות שעובדו וצריכות להכתב ליחידת האחסון. עם מילוי המאגר יועבר כל הגוש אל יחידת האחסון.

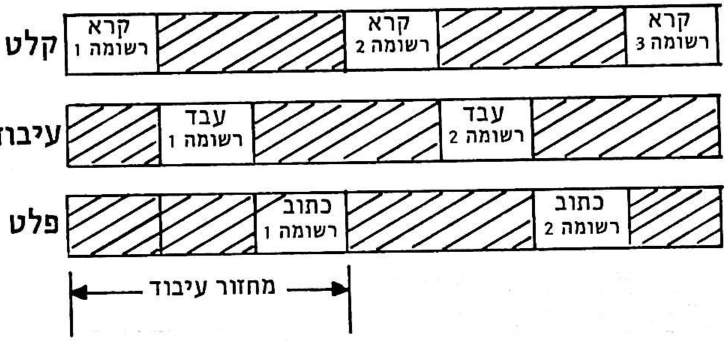


תרשים 2-32: עבודה עם מאגרי קלט/פלט נפרדים

שיטה זו מאפשרת חפיפה בין זמני העיבוד וזמני הקלט/פלט. ניתן להמשיך בעיבוד לפני שמאגר הפלט נכתב ליחידת האחסון. כפי שהוסבר, קיימים מחשבים בהם קיים מעבד המוקדש לעבודת קלט/פלט בלבד. קיימים מחשבים בהם ישנם מספר מעבדים כאלה. במערכות אלה, ניתן לקבל חפיפה אפילו בין זמני הקלט וזמני הפלט. בזמן שמעבד אחד מביא גוש חדש, מעבד אחר כותב את הגוש הקודם אל יחידת האחסון.

השיטה המוצגת בתרשים 2-32 אינה מאפשרת ליע"מ להמשיך את העיבוד, אפילו אם הגוש החדש כבר נמצא בזכרון מאחר ומאגר הפלט יכול לקבל את הרשומה המעובדת, רק לאחר שהמאגר הקודם כבר נכתב.

נדגים את משמעות החפיפה בעבודה עם שני מאגרים.



תרשים 2-33: מחזור עיבוד ללא מאגרים

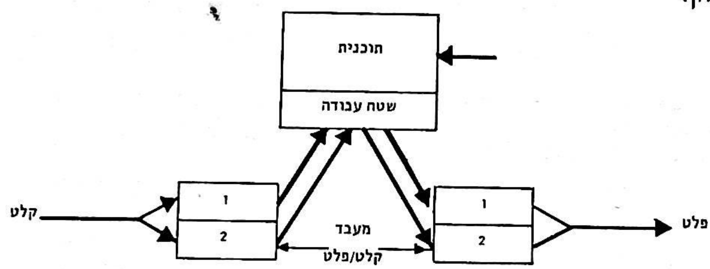
תרשים 2-33 מציג את מחזור העיבוד ללא שימוש במאגרים. השטח המקווקו מסמן שהיחידה בטלה מעבודה ומחכה לסיום הפעולה של יחידה אחרת. לדוגמא, אין אפשרות לקרוא את הרשומה השניה, עד שרשומה ראשונה לא נכתבה.



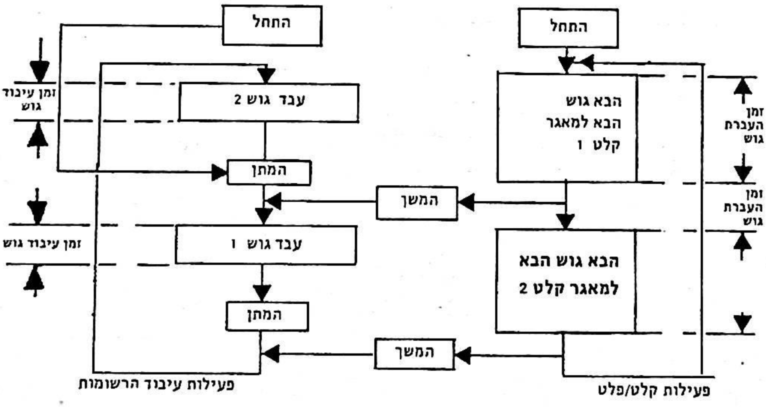
תרשים 2-34: מחזור עיבוד עם מאגרי קלט ופלט

תרשים 2-34 מציג את מחזור העיבוד עם מאגרי קלט/פלט. בזמן שקוראים לתוך מאגר הקלט את רשומה 3, ניתן לעבד את רשומה 2 ולכתוב את רשומה 1. התהליכים מתבצעים בחפיפה. נציין שלא תמיד קיימת חפיפה מלאה, מאחר וזמן העיבוד אינו מקביל בדיוק לזמן הקלט/פלט כפי שתואר בתרשים.

ליעול נוסף בתהליך העיבוד ניתן להגיע ע"י הקצאת שני מאגרי קלט/פלט לכל יחידת אחסון (Double Buffering). בזמן שקוראים לתוך מאגר קלט 1 ניתן לעבד את הרשומות הנמצאות במאגר קלט 2. עם גמר העיבוד מועברת הרשומה אל מאגר פלט 1. ברגע שמאגר פלט 1 מתמלא, מתחילים למלא את מאגר הפלט 2 ובמקביל כותבים את מאגר פלט 1 ליחידת האחסון. השימוש במאגרים נעשה לסירוגין ומאפשר איזון טוב יותר בין מהירויות העיבוד ומהירויות קלט/פלט כדי לקיים רצף עבודה של כל רכיבי המערכת וניצול טוב יותר שלהן.



תרשים 2-35: שני מאגרי קלט/פלט



תרשים 2-36: עיבוד חופף בין פעילות עיבוד וקלט/פלט

סיכום

לניהול המאגרים לקלט/פלט השפעה רבה מאד על ביצועי המחשב. הגדלת מספר המאגרים מאפשרת עיבודים חופפים בין היע"מ לבין מעבדי הקלט/פלט. הבעיה היא שריבוי מאגרים מביא לצריכת זיכרון גדולה, אך היא שקולה כנגד העלאת יעילות המערכת. כדי להתגבר על בעיות אלה, פותחו מערכות לניהול קבצים המפעילות אלגוריתמים מתוחכמים לניהול המאגרים.

2.7 שאלות

1. הסבר תפקידו של ערוץ קלט/פלט וכיצד הוא מבצע את תפקידו. האם ניתן לבנות מחשב ללא ערוץ קלט/פלט? מה תהיה צורת עבודתו?
2. הסיביות בזכרון המחשב מיצגות נתונים או פקודות. כיצד מבדיל היע"מ ביניהם?
3. אם גושים ורשומות יכולים להיות באורך משתנה, מה לגבי מאגרי הקלט/פלט? האם הם באורך משתנה? כיצד מחשבים את אורכם?
4. איזה סוג נתונים יאוחסנו בזכרון ואיזה ביחידות אחסון חיצוניות?
5. לשם מה דרוש מאגר בין יחידת האחסון לזכרון המחשב?
6. הסבר את המושגים הבאים: מרווח, גוש, מהירות העברה, צפיפות מגנוט לגבי סרטים ודיסקים.
7. קרא פירסום של יצרן המתיחס ליחידת אחסון של סרטים ודיסקים. הוצא מתוך הפירסום את כל הפרמטרים הרלוונטים.
8. חשב את מספר הכרטיסים (80 תווים) שניתן לאחסן בסרט מגנטי באורך 2400 רגל, עם מרווח של 0.6 אינץ, כאשר כותבים:
   — רשומה אחת לגוש
   — 10 רשומות לגוש.

   חשב עבור צפיפות 800 ו־1600 מסגרות לאינץ (BPI).
9. נתון סרט מגנטי באורך 2400 רגל, מרווח 0.6 אינץ, צפיפות 1600, מהירות תנועה 125 אינץ לשניה, זמן מרווח 4.8 מילישניות. הרשומות הן באורך של 200 בתים.

   חשב את הזמן הדרוש לקריאת קובץ המכיל 50,000 רשומות עבור 1, 5, 10 רשומות לגוש.
10. ככל שמספר הרשומות לגוש גדול יותר זמן קלט/פלט טוב יותר. מדוע לא לבנות גושים גדולים מאד? מה האילוצים לגודל גוש?

2.8 נספח א׳: חישוב גודל הגוש בסרט מגנטי

קיימות שיטות שונות לחישוב גודל הגוש המיטבי. לדוגמא, נציג שתי שיטות הבנויות על השיקולים הבאים:

— יעילות העיבוד

— גודל הזכרון הפנוי.

א. יעילות עיבוד

המטרה היא בנית גוש בגודל שיאפשר עיבוד חופף בין היע״מ לבין ערוץ הקלט/פלט. נחפש את גודל הגוש שזמן עיבוד הרשומות שלו שווה לזמן העברת הגוש לזכרון. לשם הפשטות, נניח שקיים רק מאגר אחד בזכרון והוא משמש גם לקלט וגם לפלט.

הקובץ מכיל רשומות באורך קבוע R, גודל הגוש B ולכן מספר הרשומות בגוש

$$b = \frac{B}{R}$$

זמן העברת רשומה מכונן לזכרון הוא R/t, לכן זמן העברת הגוש לזכרון יהיה $b \times R/t$. לזמן זה יש להוסיף את מחצית זמן האטה/האצה של הכונן ואותו נסמן ב־t_a. הזמן הכולל יהיה:

$$T_B = t_a + b \cdot \frac{R}{t}$$

נסמן ב־t_p את זמן עיבוד הרשומה. מאחר ויש b רשומות בגוש, זמן עיבוד הגוש יהיה:

$$T_p = b \cdot t_p$$

במערכת סידרתית רגילה שבה אין חפיפה בין היע״מ לבין הערוץ, זמן העיבוד הכולל של הגוש יהיה סכום זמן העברת הגוש לזכרון וזמן עיבודו.

$$T = T_B + T_p$$

במערכת עם חפיפה, העיבוד וההעברה יכולים להתבצע באותו זמן. אם T_p קטן יותר מ־T_B ניתן לבצע את העיבוד במקביל להעברת גוש חדש אל הזיכרון. סה״כ זמן העיבוד יהיה הגדול מבין זמן העיבוד וזמן ההעברה.

נסמן ב־U את שעור החפיפה.

$$u = \frac{T_P}{T_B}$$

אם $T_p > T_B$ ← $U > 1$ התהליך חסום על־ידי זמן העיבוד (CPU Bounded).

אם $T_p = T_B$ ← $U = 1$ התהליך מאוזן (Balanced).

אם $T_p < T_B$ ← $U < 1$ התהליך חסום על־ידי זמן העברת גוש (I/O Bounded).

נחשב את גודל הגוש בתהליך מאוזן, המאפשר חפיפה מלאה.

$$u = \frac{T_P}{T_B} = \frac{b \cdot t_p}{t_a + b \cdot \frac{R}{t}} = 1$$

מספר הרשומות המומלץ לגוש:

$$\boxed{b = \frac{t_a}{t_p - \frac{R}{t}}}$$

מאחר ו־t_a, R, t קבועים, ככל שזמן עיבוד הרשומה יגדל, יש להקטין את גודל הגוש לקבלת תהליך מאוזן ולהיפך. אם שווה לזמן הדרוש להעברת רשומה מכונן לזכרון, גודל הגוש שואף לאינסוף.

במציאות, חישוב גודל הגוש מורכב יותר מאחר וקיימים מספר מאגרי קלט/פלט, מתבצע עיבוד של מספר קבצים ומשתמשים בעיבוד מרובב (Multiprogramming) שבו קיימת בכל מקרה חפיפה מסוימת בין העיבוד לפעולות קלט/פלט.

ב. גודל זיכרון פנוי

השיטה הקודמת לא התחשבה בכמות הזכרון העומדת לרשותנו למאגרים. אם זמן עיבוד הרשומה קטן מאד, לשמירת תהליך מאוזן יש לבנות גושים גדולים מאד. גושים גדולים מחייבים הקצאת מאגרים גדולים בזיכרון. אם התוכנית מעבדת בו זמנית מספר קבצים, הבעיה מחריפה.

נציג שיטה לחישוב גודל הגוש על פי שיקולי כמות זיכרון פנויה. גודל הגוש יחושב מתוך הנחה שהתוכנית מעבדת n קבצים שונים ולרשות התוכנית עומד גודל זיכרון S (יש להדגיש שהנושא רלוונטי בעיקר במחשבים קטנים וזעירים בהם לא קיימת שיטת זיכרון בפועל — Virtual Storage).

נשתמש בסימונים הבאים:

N_i — מספר הרשומות בקובץ i,

R_i — אורך רשומה בקובץ i בתווים, בהנחה שהרשומות הן באורך קבוע,

b_i — מספר הרשומות לגוש בקובץ i,

S — שטח הזכרון הכולל הפנוי למאגרים.

המטרה היא בנית מספר גושים מינימלי בכל הקבצים לקבלת ניצולת אחסון גבוהה. מאחר ובקובץ יש N רשומות ומספר הרשומות לגוש הוא b, מספר הגושים לקובץ יהיה N/b. נדרוש שמספר הגושים לכל הקבצים יהיה מינימלי.

$$\text{Min} = \left\{ \sum_{i=1}^{n} \frac{N_i}{b_i} \right\}$$

קיים אילוץ ששטח המאגרים לא יעלה על S. גודל המאגר הדרוש לכל קובץ הוא מכפלת מספר הרשומות בגוש בגודל הרשומה.

$$\sum_{i=1}^{n} b_i R_i \leqslant S$$

ניתן לפתור בעית אופטימיזציה זו תוך שימוש בכופלי לגרנז'. לפי Walters מספר הרשומות האופטימלי בקובץ i יהיה:

$$b_i^{opt} = \frac{S \cdot \sqrt{\frac{N_i}{R_i}}}{\sum_{j=1}^{n} \sqrt{N_j R_j}}$$

מאחר ומספר הרשומות לגוש חייב להיות מספר שלם, יש לעגל את התוצאה המתקבלת. העיגול גורם לסטיה מהאופטימום.

<u>דוגמא:</u> נדגים שיטת חישוב הגוש. תוכנית מעבדת שלשה קבצים הנמצאים על סרטים מגנטים.

קובץ 1: מכיל 100,000 רשומות באורך 1000 תווים

קובץ 2: מכיל 10,000 רשומות באורך 500 תווים

קובץ 3: מכיל 1,000 רשומות באורך 80 תווים

שטח זיכרון למאגרים 12,000 תווים. נשווה את מספרי הגושים, זמני העיבוד ושטח האיחסון עבור שלשה מקרים:

- רשומה אחת לגוש,
- חלוקה שווה של השטח לכל אחד מהקבצים,
- חישוב גודל הגוש האופטימלי על סמך הנוסחה.

<u>רשומה אחת לגוש:</u>

קובץ 1: מספר גושים = 100,000 $b_1 = 1$

קובץ 2: מספר גושים = 10,000 $b_2 = 1$

קובץ 3: מספר גושים = 1,000 $b_3 = 1$

סה"כ 111,000 גושים בשלשת הקבצים.

<u>חלוקה שווה:</u> נקצה לכל אחד משלשת הקבצים 4000 תווים בזכרון.

מספר גושים $= \frac{100.000}{4} = 25.000$ $b_1 = \frac{4000}{1000} = 4$

מספר גושים $= \frac{10.000}{8} = 1250$ $b_2 = \frac{4000}{500} = 8$

מספר גושים $= \frac{1000}{50} = 20$ $b_3 = \frac{4000}{80} = 50$

סה"כ קיבלנו 26,270 גושים בשלשת הקבצים.

חישוב גודל גוש אופטימלי: נשתמש בנוסחה לחישוב גודל הגוש האופטימלי:

$$\sum_{j=1}^{3} \sqrt{N_j R_j} = \sqrt{100.000 \times 1000} + \sqrt{10.000 \times 50} + \sqrt{1000 \times 80} = 12.519$$

$$b_1 = \frac{12.000}{12519} \sqrt{\frac{100.000}{1000}} = 9.59 \qquad \text{מספר גושים} = \frac{100.000}{9.59} \cong 10427$$

$$b_2 = \frac{12.000}{12519} \sqrt{\frac{10.000}{500}} = 4.29 \qquad \text{מספר גושים} = \frac{10.000}{4.29} \cong 2375$$

$$b_3 = \frac{12000}{12519} \sqrt{\frac{1000}{80}} = 3.39 \qquad \text{מספר גושים} = \frac{1000}{3.39} \cong 294$$

הפתרון האופטימלי מכיל 13,062 גושים. מאחר ומספר הרשומות לגוש חייב להיות שלם נעגל את התוצאות: $b_1 = 9;\ b_2 = 4;\ b_3 = 3$

במקרה זה מספר הגושים בשלשת הקבצים יהיה 13,946 שהוא גבוה מהאופטימום אבל עדיין נמוך בהרבה משני המקרים הקודמים.
נסכם את התוצאות שהתקבלו בטבלה:

	מספר רשומות	אורך רשומה	מספר גושים: רשומה לגוש	חלוקה שווה	חישוב מעוגל
קובץ 1	100,000	1000	100,000	25,000	11,112
קובץ 2	100,000	500	10,000	1,250	2,500
קובץ 3	1,000	80	1,000	20	334
סה"כ מספר גושים			111,000	26,270	13,946

על סמך הנוסחאות שפיתחנו לחישוב קיבולת הסרט וזמן עיבוד, נחשב את שטח האחסון הכולל וזמן העיבוד הדרושים עבור כל אחת מהשיטות.
לשם כך נשתמש בנתונים הבאים:

צפיפות המיגנוט = 1,600 תווים לאינץ
מהירות העברה = 40,000 תווים לשניה
זמן האטה/האצה = 20 מילישניות
גודל מרווח = 0.6 אינץ

על סמך נוסחת הקיבולת של סרט ניתן לחשב את השטח הכולל הדרוש לאחסון הקובץ עבור כל אחד מהמקרים:

$$\text{שטח אחסון באינץ} = \text{מספר גושים} \left(\frac{\text{מספר תווים לגוש}}{\text{צפיפות מגנוט}} + \text{גודל מרווח} \right)$$

זמן עיבוד כל הקובץ יהיה זמן עיבוד הגוש × מספר הגושים

זמן עיבוד = מספר גושים (מחצית זמן האטה/האצה + $\frac{\text{מספר תווים לגוש}}{\text{מהירות העברה}}$)

נניח שזמן עיבוד הרשומה בזכרון זניח.

	רשומה לגוש		חלוקה שווה		חישוב מעוגל	
	שטח אחסון	זמן עיבוד	שטח אחסון	זמן עיבוד	שטח אחסון	זמן עיבוד
קובץ 1	122,500	2,600	77,500	2,525	69,172	2,511
קובץ 2	9,125	135	3,875	126	4,625	127
קובץ 3	650	3	62	2	250	2
סה"כ	132,275	2,738	81,437	2,653	74,047	2,640

תוצאה:

— שטח האחסון הכולל בשיטת החישוב קטן מאשר בשיטות ההקצאה של רשומה לגוש ושיטת החלוקה השווה.

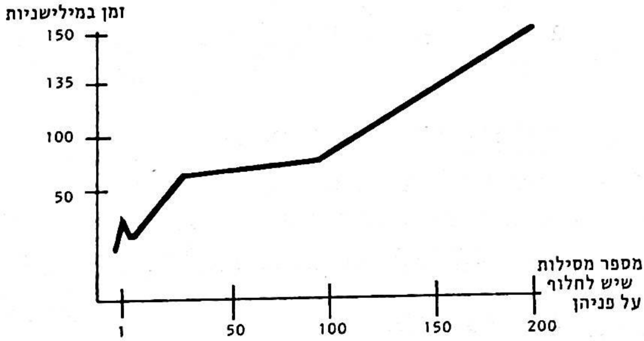
— זמן העיבוד הכולל בשיטת החישוב ובשיטת החלוקה השווה זהה כמעט ונמוך מאשר בשיטת ההקצאה של רשומה לגוש.

2.9 נספח ב': חישוב זמן חיפוש בדיסק

מבין שלשת המרכיבים של זמן הגישה לנתונים בדיסק זמן החיפוש הוא הגדול ביותר. בנספח זה נרחיב את הדיון על זמני חיפוש.

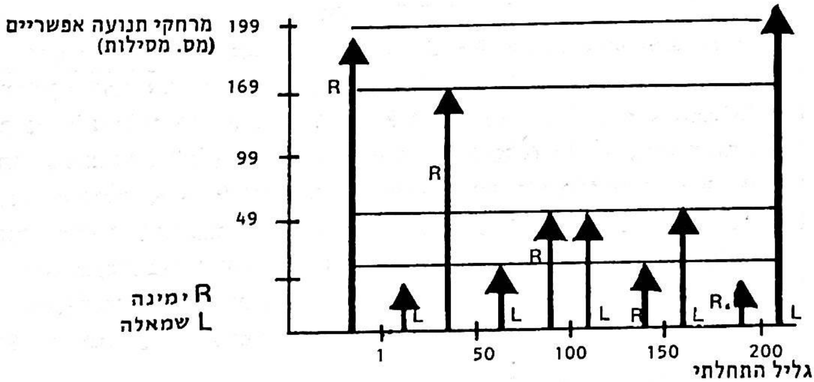
הגדרה: זמן חיפוש הוא הזמן הדרוש למיקום ראשי קריאה/כתיבה במקום הדרוש לקריאה או לכתיבה.

זמן החיפוש תלוי במסילה בה נמצא ראש הקריאה/כתיבה ובמסילה אליה עליו להגיע. ככל שהמרחק ביניהן גדול יותר, זמן החיפוש יהיה גדול יותר.



תרשים 2-37: זמני חיפוש עבור דיסק 2314 של יבמ

כפי שניתן לראות מתרשים 2-37 לא קיים יחס לינארי פשוט בין מספר המסילות שיש לחלוף על פניהן לבין הזמן הדרוש לביצוע תנועה זו.



תרשים 2-38: מספר הגלילים שיש לחלוף כפונקציה של מקום מנוחת הזרוע

חישוב מרחק חיפוש ממוצע: נניח שקובץ מכיל N גלילים. החיפוש מתבצע בין גליל המקור (המקום בו נח הראש) וגליל היעד (הגליל אליו יש להניע את הראש). גליל המקור יכול להיות כל אחד מ־i הגלילים (i = 1, 2, ..., N). גליל היעד יכול להיות כל אחד מ־j הגלילים (j = 1, 2, ..., N).

המרחק בין גליל מקור i לגליל יעד j יהיה |i - j|. נחשב את סכום המרחקים מכל גליל לכל גליל ונחלק במספר המרחקים האפשריים לשם קבלת מרחק החיפוש הממוצע. סכום המרחקים האפשריים:

$$\sum_{i=i}^{N}\ \sum_{j=i}^{N}(i-j) = \sum_{i=i}^{M}\ 2\sum_{j=1}^{i}(i-j) =$$

$$2\sum_{i=i}^{M}\left(i^2-\frac{i(i+1)}{2}\right) = \sum_{i=i}^{N}(i^2-i) =$$

$$\frac{N(N+1)\ (2N+1)}{6}-\frac{N(N+1)}{2} = \frac{N(N+1)\ (N-1)}{3}$$

קיימים N(N - 1) צירופים בין גליל התחלתי לגליל יעד. נחלק את סכום המרחקים האפשריים במספר הצירופים:

$$\frac{N(N+1\ \ (N-1)}{3N(N-1)} = \frac{N+1}{3} \cong \frac{N}{3}$$

קיבלנו שהמרחק הממוצע שהזרוע עוברת הוא כשליש ממספר הגלילים שהקובץ תופס בדיסק. המסקנה היא שאומדן טוב לזמן חיפוש ממוצע יהיה הזמן הדרוש כדי לחלוף על שליש מהגלילים שהקובץ תופס. נציין שחישוב זה נכון עבור הסתברות גישה זהה לכל אחד מהגלילים.

חישוב זמן חיפוש ממוצע: ניתן לחשב את זמן החיפוש הממוצע בצורה מדויקת יותר את הסתברות הגישה לגליל זהה עבור כל הגלילים. אם הקובץ תופס N גלילים, חיפוש שמאלה של K גלילים (K = 1, 2, ..., N) אפשרי אם הזרוע נמצאת בגליל K + 1 ואילו חיפוש ימינה של K גלילים אפשרי אם הזרוע נמצאת בגליל N-K. לדוגמא, חיפוש שמאלה של שני גלילים, K = 2, אפשרי אם הזרוע נמצאת בגליל K + 1 = 3. חיפוש ימינה של שני גלילים אפשרי אם הזרוע נמצאת בגליל 198 (בהנחה שקיימים 200 גלילים במשטח המַגנוט).

מאחר וקיימים סה"כ N(N-1) צירופים אפשריים בין גליל המקור לגליל יעד וקיימת הסתברות שווה לכל צירוף נקבל:

$$\text{הסתברות גישה לגליל} = \frac{1}{N(N-1)}$$

התוחלת של המרחקים שנחלוף תהיה:

$$S_i = \underbrace{\sum_{K=i+1}^{N} \frac{1}{N(N-1)}}_{\text{שמאלה}} + \underbrace{\sum_{K=1}^{N-i} \frac{1}{N(N-1)}}_{\text{ימינה}} = 2\frac{N-i}{N^2-N}$$

$$i = 1, 2, \ldots N-1$$

S_i הוא תוחלת המרחק שיש לעבור אם הזרוע נמצאת בגליל מקור i.

לחישוב זמן החיפוש הממוצע, יש להכפיל את המרחק בזמן הדרוש לעבור את הגלילים.

$$T = \sum_{i=0}^{N-1} t_i \; S_i = 2\sum_{i=0}^{N-1} t_i \frac{N-i}{N^2-N}$$

נציין שההנחה של הסתברות גישה זהה לכל גליל אינה מתקיימת בדרך כלל במציאות. התרכזות הפעילות סביב גלילים מסוימים יכולה לשפר את זמן הגישה. ניתן גם להשתמש בנוסחה מקורבת של זמן החיפוש. נוסחה זו תלויה בכונן וניתנת ע"י היצרן. המיבנה הכללי שלה $T=a+bK$ כאשר a ו-b קבועים כלשהם ו-K מספר הגלילים שיש לחלוף, מאחר ובממוצע $K=N/3$ נקבל קירוב לזמן חיפוש $T=a+bN/3$.

פרק שלישי
תכנון רשומות

3.1 מבוא
3.2 איפיון רשומות
3.3 מפתחות
3.4 קביעת השדות המרכיבים את הרשומה
3.5 אירוע — תכנון רשומות
3.6 שאלות

בפרק זה נציג את השיקולים השונים בתכנון רשומות ויוסברו המאפיינים השונים שלהן.

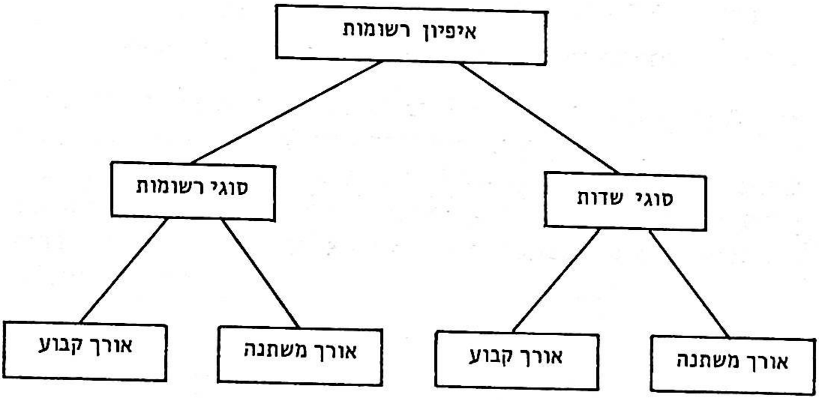
3.1 מבוא

יחידת המידע החשובה ביותר בעיבוד נתונים היא **הרשומה**. כל פעולות הקלט/פלט של תוכנית המחשב מתיחסות לרשומה. תכנון טוב של הרשומה הוא הבסיס להפעלה טובה של מערכת מידע ממוחשבת. שיטות הארגון המתקדמות משתמשות במפתח הרשומה לאיתורה המהיר ביחידת האחסון. בפרק זה נסקור את המאפינים השונים של הרשומות והמפתחות שלהן ונציג שיקולים אחדים המנחים את מנתח המערכות בקביעת מבנה הרשומה.

3.2 איפיון רשומות

הגדרה: רשומה הינה **אוסף של שדות בעלי משמעות לוגית**. השדות מכילים את יחידות המידע הבסיסיות של פרט מסויים באוכלוסיה. לפיכך, הרשומה היא אוסף הנתונים של פרט כלשהו מזוהה באוכלוסיית הקובץ.

התכונה החשובה ביותר של הרשומה והשדות המרכיבים אותה, מנקודת מבט המערכת לניהול קבצים, היא האורך שלהם.



תרשים 3-1: איפיון רשומות

סוגי שדות

מבחינים בין שדות בעלי אורך קבוע ושדות בעלי אורך משתנה.

שדות בעלי אורך קבוע: שדות אלה מורכבים ממספר קבוע של תווים. לדוגמא, מספר קטלוגי המכיל 9 תווים נומריים. בכל הרשומות נמצא 9 ספרות בשדה המספר הקטלוגי. הטיפול בסוג זה של שדות הינו פשוט יחסית.

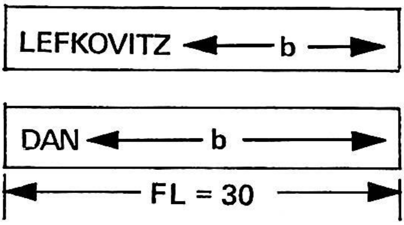
שדות בעלי אורך משתנה: שדות אלה מורכבים ממספר משתנה של תווים. לדוגמא כתובת או שם משפחה. קיימות כתובות בעלות 15 תווים ואחרות בעלות 40 תווים. טווח השינוי יכול להיות שונה ממקרה למקרה. קיימות מספר שיטות לטיפול בשדות בעלי אורך משתנה:

א. יישור (Justifying): בשיטה זו השדות בעלי האורך המשתנה מיוצגים במחשב כשדות בעלי אורך קבוע. גודל השדה בעל האורך הקבוע יהיה בגודל המירבי שהשדה המשתנה יכול לקבל. הנתונים מועברים לשדה באורך קבוע תוך יישור השדה לאחד הכיוונים (ימין או שמאל) והשלמת התווים הנותרים לערך מוסכם מראש. היישור מתבצע בהתאם לסוג הנתונים:

- נתונים מספריים — יישור ימינה והשלמת תווים מובילים באפסים,
- נתונים אלפבתיים — יישור שמאלה באנגלית ויישור ימינה בעברית, והשלמת התווים הסופיים בתווי רווח (Blank, Space).

דוגמא: שדה שם העובד הוא שדה אנגלית, בעל אורך משתנה. נבנה אותו באורך 30 תווים ונישר את הנתונים לשמאל. שם עובד ארוך יותר יקוצץ.

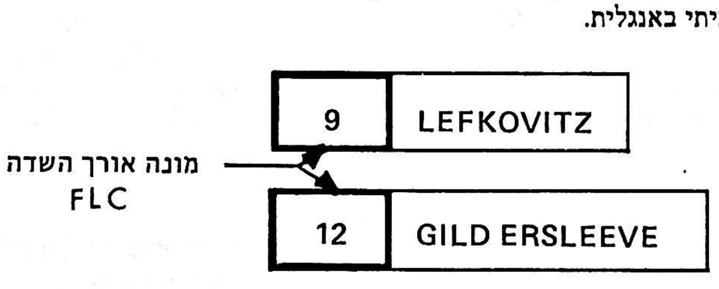
אורך שדה FL — Field Lenght.



תרשים 3-2: שדה באורך קבוע עם תוכן באורך משתנה

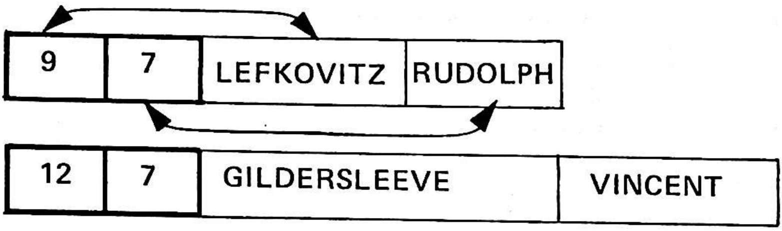
כפי שניתן לראות יש בשיטה זו בזבוז בשטח אחסון, אבל מאידך היא פשוטה מאד מאחר והטיפול הוא בשדות בעלי אורך קבוע.

ב. מונה אורך השדה (Field Lenght Count): אם השדה משתנה באורכו בצורה קיצונית שיטת היישור היא בזבזנית מאד. ניתן לפתור בעיה זו על־ידי הוספת שדה אחד נוסף המכיל את אורך השדה. נשתמש שוב בדוגמא של שדה אלפביתי באנגלית.



תרשים 3-3: שימוש במונה אורך שדה

הטיפול בשדות אלה מורכב יותר. יש לפענח תחילה את תוכן שדה המונה ורק אחר כך לטפל בשדה הנתון עצמו. אם הרשומה מכילה מספר שדות בעלי אורך משתנה מקובל לרכז יחדיו את כל שדות המונה.



תרשים 3-4: מספר שדות מונה אורך

השימוש בשדות בעלי אורך משתנה גורם לרשומה כולה להיות בעלת אורך משתנה ולכן הטיפול בה מורכב יותר. יש לזכור ששדה המונה עצמו תופס מקום ומתווסף לאורך הנתונים שבגוף הרשומה.

ג. <u>סימון שדות:</u> ניתן להשתמש בסימן מיוחד המפריד בין שדות בעלי אורך משתנה. סימן מיוחד זה אינו יכול להופיע בתוכן השדות. הטיפול ברשומות מסוג זה מסורבל יותר מאחר ויש לבצע סריקה של כל השדות ולחפש את הסימן המיוחד המפריד ביניהם.



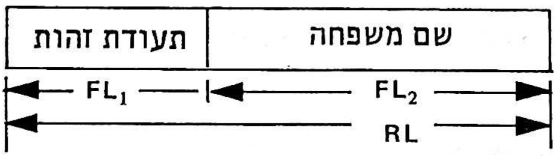
תרשים 3-5: סימון מיוחד בין שדות בעלי אורך משתנה

<u>סוגי רשומות</u>

מבחינים בין רשומות באורך קבוע ורשומות באורך משתנה.

<u>רשומות באורך קבוע:</u> אם כל השדות המרכיבים את הרשומה הם בעלי אורך קבוע גם הרשומה תהיה בעלת אורך קבוע.

אורך רשומה RL — Record length



תרשים 3-6: רשומה בעלת אורך קבוע

אורך הרשומה כולה מחושב כסכום כל השדות המרכיבים אותה:

$$RL = \sum_{i=1}^{n} FL_i$$

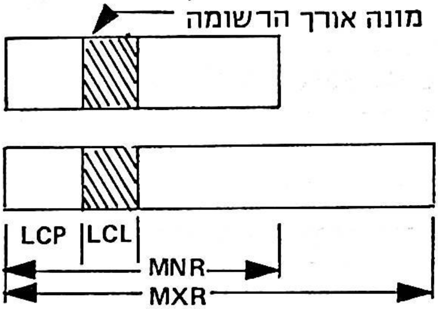
n — מספר שדות ברשומה

דוגמא ליישום סוג זה של רשומות בקובול:

```
FD  OVDIM-FILE
    RECORD CONTAINS 53 CHARACTERS
    DATA RECORD IS OVDIM-REC.
01  ODVIM-REC.
    03 ZEHUT          PIC 9(8).
    03 SHEM.
       05 PRATI    PIC X(15).
       05 MISHPAHA  PIC X(30).
```

<u>רשומות באורך משתנה:</u> כאשר חלק מהשדות המרכיבים את הרשומה הם בעלי אורך משתנה או קיימים כמה סוגי רשומות באותו קובץ נקבל רשומות באורך משתנה. קיימות מספר שיטות לניהול רשומות באורך משתנה:

<u>א. מונה אורך רשומה (Record Lenght Count):</u> בשיטה זו מוסיפים לרשומה שדה אחד נוסף המציין את אורך הרשומה. שדה זה הוא חלק מהרשומה ותוכנו באחריות המשתמש. יש להגדיר את אורך הרשומה המינימלי והמקסימלי כדי שהמערכת לניהול קבצים תוכל להקצות את המאגרים הדרושים. שדה המונה חייב להמצא בכל הרשומות באותו מקום.



LCP — Length Count Position
LCL — Length Count Length
MNR — Minimal Record Length
MXR — Maximal Record Length

MNR ⩾ LCP + LCL

תרשים 3-7: רשומות באורך משתנה עם שדה מונה אורך

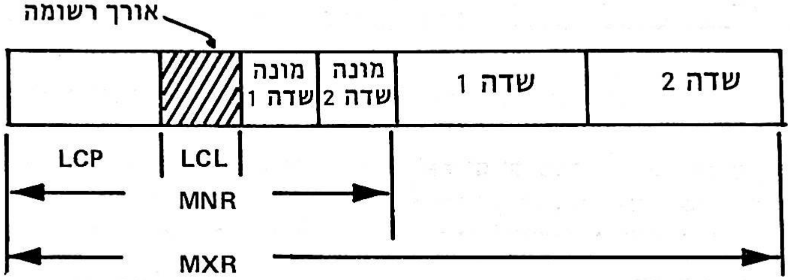
המערכת לניהול קבצים "יודעת" את אורך הרשומה מתוך ידיעת הפרמטרים LCL, LCP ופענוח תוכן שדה המונה.

```
FD X-FILE
   RECORD CONTAINS 10 TO 100 CHARACTERS
   DEPENDING ON ORECH
   DATA RECORD IS X-REC.
   01   X-REC.
        03  DATA-1              PIC X(8).
        03  ORECH               PIC 99.
        03  DATA-2              PIC X(90).
```

הפרמטרים של רשומה זו יהיו:

.MXR = 100 ,MNR = 10 ,LCL = 2 ,LCP = 9

שיטה זו מאפשרת ניהול שדות בעלי אורך משתנה, כפי שמודגם בתרשים 3-8.



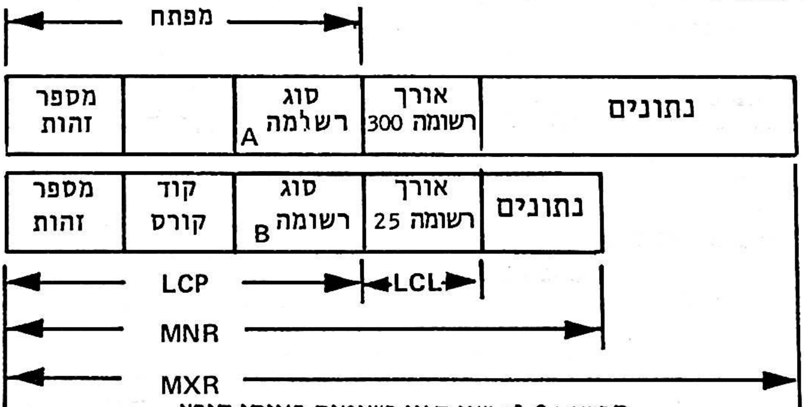
תרשים 3-8: דוגמא לשימוש בשדות משתנים

בדוגמא המופיעה בתרשים 3-8, הרשומה מכילה שני שדות בעלי אורך משתנה. לכל אחד משדות אלה יש שדה מונה אורך. בנוסף יש לרשומה שדה מונה אורך רשומה. המתכנת צריך לחשב את אורך הרשומה על־ידי סיכום התוכן של שדות מונה השדות והחלק הקבוע של הרשומה. נשים לב ששדות המונה מופיעים בכל הרשומות באותם מקומות ולכן הם כלולים ב־MNR.

שיטה זו מאפשרת טיפול במספר סוגי רשומות שונות במסגרת אותו הקובץ. נציג לדוגמא קובץ שבו קיימים שני סוגי רשומות:

— רשומת סטודנט המכילה פרטים אישיים ואורכה 300 בתים.
— רשומת ציונים המכילה את ציוני הבחינות שהוא עבר ואורכה 25 בתים.

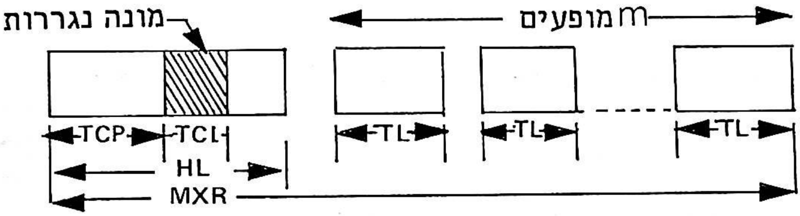
כדי לזהות את שני סוגי הרשומות נוסיף לכל רשומה שדה נוסף הנקרא "סוג רשומה".



תרשים 3-9: שני סוגי רשומות באותו קובץ

המפתח של הרשומות בתרשים 3-9 הוא באורך LCP. לקריאת רשומת פרטים אישיים נעביר את מספר הזהות וסוג רשומה A למפתח. אם רוצים לקרוא את רשומת הציונים נעביר מספר זהות, קוד הקורס וסוג רשומה B. זו דוגמא לשיטה שבה שתי רשומות זרות ובעלות מבנה שונה לחלוטין מופיעות באותו קובץ.

ב. **רשומה עם נגררות** (Trailer): לעיתים קבוצת נתונים שיש לנהל בקובץ היא בעלת מספר משתנה של מופעים. לדוגמא, במערכת לניהול מלאי פריט כלשהו יכול להופיע במספר מחסנים. בכל מחסן יש לרשום את הכמות במלאי ותאריך הכניסה למחסן. דוגמא נוספת, ברשומת עובד יש לנהל מידע על הילדים של העובד. לכל ילד יש לרשום את השם, תאריך לידה, מין ועוד. מאחר ויתכנו עובדים ללא ילדים ואילו לאחרים יש מספר ילדים שונה אין זה נוח לנהל את הנתונים האלו בשדות בעלי אורך קבוע. נוח מאד לרשום נתונים במבנה זה בצורת ניגררות בעלות אורך קבוע אבל במספר משתנה של מופעים.



TCP — Trailer Count Position
TCL — Trailer Count Length
TL — Trailer Length
HL — Header Length

תרשים 3-10: רשומה עם נגררות

התוכן של שדה מונה הנגררות מכיל את מספר הנגררות המופיעות ברשומה. המערכת לניהול קבצים תחשב את אורך הרשומה על פי פרמטרים אלה. לקביעת גודל המאגר הדרוש על המתכנת לציין את מספר הנגררות המינימלי והמקסימלי.

דוגמא: יש לנהל את הציונים של הסטודנט כמו בדוגמא הקודמת. מספר הציונים המקסימלי הוא 30 (מספר המקצועות שנבחן בהם).



תרשים 3-11: דוגמא — ניהול רשומות ציונים לסטודנט

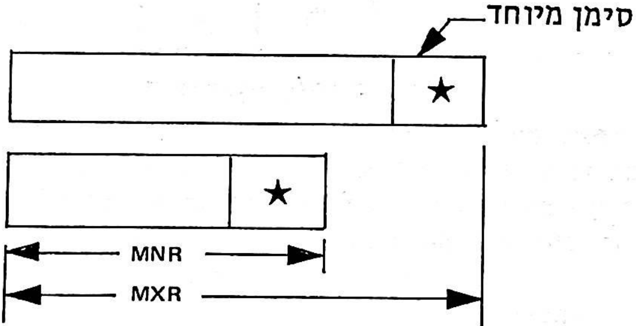
```
FD  STUDENT-FILE
    RECORD CONTAINS 80 TO 440 CHARACTERS
    DATA RECORD IS STUDENT-REC.
01  STUDENT-REC.
    03 ZEHUT                               PIC 9(8) .
    03 SHEM                                PIC X(30) .
    03 KTOVET                              PIC X(40) .
    03 MONE-NIGRERET                       PIC 99 .
    03 NIGRERET OCCURS 0 TO 30
           DEPENDING ON MONE-NIGRERET.
       05 KOD-KURS                         PIC 9(3).
       05 TARICH                           PIC 9(6).
       05 TZIUN                            PIC 9(3).
```

הפרמטרים של רשומה זו יהיו: TCP = 79, TCL = 2, HL = 81, TL = 12, MNR = 80, MXR =440.

$$MXR = HL + \left(\text{מספר נגררות מקסימלי}\right) * TL$$

הבעיה בשיטת הנגררות היא שאם הנגררות ארוכות או כאשר יש מספר גדול מאד של נגררות אפשריות הרשומות יכולות להיות ארוכות מאד והטיפול בהן באחסון ובעיבוד אינו נוח.

ג. סימון סוף רשומה: בדומה לשיטה שהוצגה לסימון שדות בעלי אורך משתנה, ניתן להשתמש בסימון מיוחד גם לציון סוף הרשומה (Record-Mark).



תרשים 3-12: רשומות עם סימון סוף רשומה

הסימן שנבחר לסמן את סוף הרשומה אינו יכול להופיע בתוכן הרשומה. לאיתור אורך הרשומה, סורקת המערכת לניהול קבצים את כל הרשומה ומחפשת את הסימן המיוחד. שיטה זו אינה מומלצת מאחר והיא יקרה מאד מבחינת צריכת זמן העיבוד של היע"מ.

3.3 מפתחות (Record Key)

הגדרה: המפתח הינו שדה, תת־שדה או קבוצה של שדות המזהים בצורה חד משמעית את הרשומה.

מבחינים בין שני סוגי מפתחות:

— <u>מפתח סדר (Order Key):</u> משמש לקביעת מיקום הרשומה בקובץ. מקובל לקרוא לו מפתח ראשי. בארגון סידרתי מיקום הרשומה אינו נקבע לפי מפתח (אלא אם כן ממיינים את הקובץ תחילה) לכן נהוג לומר שזהו ארגון ללא מפתח. בשאר שיטות הארגון המשתמשות ביחידות אחסון בגישה ישירה, המפתח משמש לקביעת מיקום הרשומה בקובץ.

— <u>מפתח חיפוש (Search Key):</u> משמש לאיתור הרשומה בקובץ. אם הוא שונה ממפתח הסדר, נוהגים לקרוא לו מפתח משני. לכל קובץ בגישה ישירה יש מפתח סדר אחד ומפתח חיפוש אחד או יותר.

דוגמא: קובץ עובדים שבו השדות ברשומה הם מספר זהות, שם עובד, מספר מחלקה.

מפתח משני ← מחלקה

מפתח ראשי ← זהות

מחלקה	שם	זהות
א'	משה	824
ב'	יצחק	915
א'	רן	969

תרשים 3-13: מבנה קובץ עובדים

השדה המזהה כל רשומה בקובץ בצורה חד־משמעית יהיה תעודת זהות. נבחר בו כמפתח סדר ומפתח חיפוש. ניתן גם לבקש מידע על כל העובדים השייכים למחלקה מסוימת. במקרה זה יש להגדיר את מספר המחלקה כמפתח חיפוש. החיפוש של רשומות לפי מפתח שאינו מפתח הסדר יוסבר בפרק על אינדקסים משניים ומבנים משורשרים.

נוהגים להבחין בשני מצבים שהערכים של המפתח יכולים לקבל:

<u>מפתח יחודי (Unique Key):</u> המפתח מזהה בצורה חד־משמעית רשומה אחת בלבד. בדוגמא הקודמת תעודת הזהות מזהה בצורה חד־משמעית רשומה בקובץ. לכל עובד יש מספר תעודת זהות שונה ולכן הערך שיופיע ברשומה יהיה שונה. מפתח הסדר חייב להיות יחודי ברוב שיטות הארגון המשתמשות ביחידות אחסון בגישה ישירה (אינדקס סידרתי וארגון ישיר).

מפתח כפול (Duplicate Key): המפתח אינו מזהה בצורה חד־משמעית רשומה אחת אלא קבוצה של רשומות. בדוגמא קודמת, המחלקה הינו מפתח מסוג זה. מאחר ומספר עובדים שייכים לאותה מחלקה ירשם מפתח זה בשדה המחלקה של כל רשומות העובדים באותה מחלקה.

קביעת המפתחות של הרשומה הינם פונקציה של היישום ואפשרויות הגישה לנתונים שיש להעמיד לרשות המשתמש. בכל אופן, אין לבנות מפתחות משניים רבים מדי מאחר והטיפול בהם הינו יקר מבחינת משאבי המחשב. יש להגדיר מפתח כמפתח משני רק כאשר ברור שיש לגשת לרשומות בסדר שונה מאשר קובע מפתח החפוש של הרשומות.

לפעמים רוצים לחסוך במקום ברשומה ואז ניתן להשתמש בטבלאות עזר חיצוניות. לדוגמא אם ברשומה מופיע מספר מחלקה אין צורך שנחזיק בתוך הרשומה גם את שם המחלקה. ניתן לשמור טבלה חיצונית המבצעת תרגום בין מספר המחלקה לשם המחלקה. אם נרצה לגשת לרשומה לפי שם המחלקה יש לגשת תחילה לטבלה, לבצע תרגום למספר מחלקה ולגשת לקובץ. זוהי דוגמא לחיפוש רשומה לפי שדה שאינו נמצא ברשומה ולכן נקרא לו **מפתח חיצוני**. שיטה זו מחייבת גישה נוספת לטבלה אבל מאידך מפשטת את תהליך העדכון. שינוי שם מחלקה למשל, מתבצע במקום אחד בלבד ואינו מחייב עדכון כל רשומות הקובץ שבו מופיע מספר המחלקה.

3.4 קביעת השדות המרכיבים את הרשומה

הרשומה מייצגת פרט כלשהו באוכלוסיה שעליו אוספים נתונים. לדוגמא במערכת לניהול מלאי, רשומות המלאי יכילו פרטים רלוונטיים בנושאי המלאי: מספר קטלוגי, תאור פריט, יחידת מידה, כמות. השדות המרכיבים את הרשומה נקבעים על פי מטרת המערכת ושיקולי יעילות. נשתמש בדוגמא לתכנון רשומה לניהול עובדים במפעל. הרשומה תכיל את הפרטים הבאים: פרטים אישיים, כתובת, נתוני השכלה, נסיון מקצועי, קורסים. אם כעבור זמן מה נתבקש לשלוף את רשומת העובד הגר בבית הגדול ביותר, לא נוכל לעשות זאת בעזרת המערכת הממוחשבת מכיון שפרט זה אינו מופיע בנתוני הכתובת. כאשר הפרטים הדרושים לקבלת המידע אינם מופיעים ברשומה אי אפשר לקבל מידע זה. במקרה שלפנינו הנתון על גודל הבית (מספר דירות למשל) לא הוגדר מראש בין הנתונים הנדרשים.

בגלל השוני בתכלית המערכות, יתכן מצב שבו שתי מערכות לניהול עובדים בשני מפעלים יכילו שדות שונים, אם מפעל אחד עוסק בייצור חומר רדיואקטיבי יש לשער שנמצא ברשומה פרטים אודות בריאות העובדים והבדיקות הרפואיות שלהם. במפעל העוסק בייצור בגדים יש לשער שהנושאים הבריאותיים לא יבואו לידי ביטוי באותה מידה ברשומה.

סגמנטציה של רשומות

עד עכשיו התיחסנו לרשומות כאל אוסף של שדות בעלי משמעות לוגית. לא הכרנו מצב של רמת ביניים בין הרשומה לבין השדה. ברמה זו ניתן להגדיר את אוסף השדות המתיחסים לנושא מסוים בתוך הרשומה. אוסף שדות זה מהווה חלק של הרשומה הלוגית שבה דנו קודם.

נשתמש ברשומות העובד מהדוגמא הקודמת:

— פרטים אישיים כוללים: מספר זהות, שם, כתובת, טלפון;
— פרטים דמוגרפיים: שנת לידה, ארץ לידה, שנת עליה, מין;
— נתוני השכלה: סוג השכלה, מספר שנים, תעודה;
— נתוני קורסים: קוד הקורס, תאריך הקורס, מוסד, ציון.

עד כה הנחנו שאת כל הפרטים האלה השייכים לפרט אחד באוכלוסיה מחזיקים בתוך אותה רשומה במבנה אשר מקובל לכנות אותו **רשומה שטוחה** (Flat Record).

קורסים			השכלה	פרטים דמוגרפיים	פרטים אישיים
m	----	1			

תרשים 3-14: רשומת עובד "שטוחה"

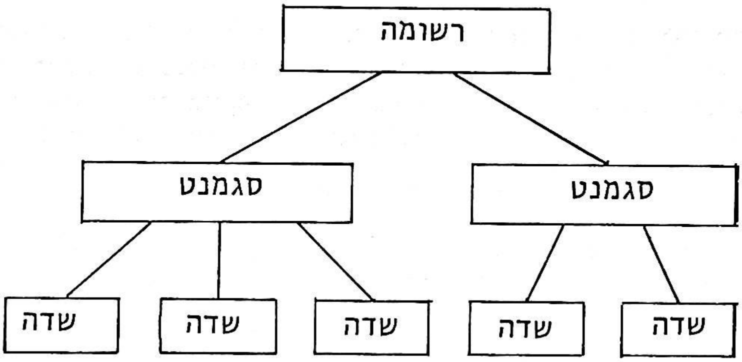
שיטה זו יוצרת שתי בעיות:

— רשומות ארוכות מאד. כל תוכנית מקבלת את כל הרשומה גם אם רוב הנתונים אינם דרושים לה. לדוגמא, תוכנית המפיקה דוח על התפלגות העובדים לפי גיל אינה זקוקה לפרטים על השכלת העובד או הקורסים שהשתתף בהם. הרשומות הארוכות גורמות ליצירת גושים גדולים ולבזבוז בהעברת הנתונים בין יחידות האחסון לזיכרון המחשב.

— טיפול לא נוח בנתונים בעלי מספר משתנה של מופעים. לדוגמא, ניתן לשמור ברשומה מספר מוגבל של קורסים ונתוני השכלה בגלל שיקולים של מבנה הרשומה (אורך קבוע למשל) או גודל הגוש.

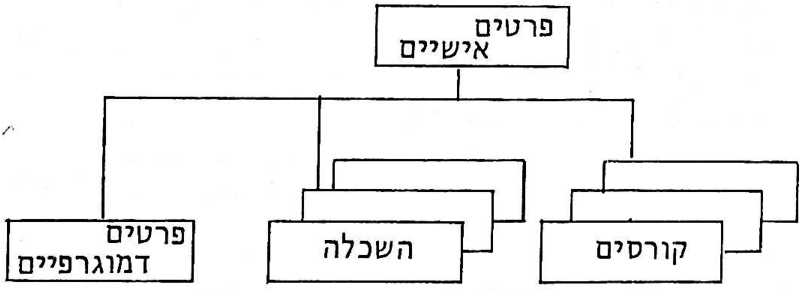
פתרון לבעיות אלה נמצא על־ידי **סגמנטציה של הרשומות.**

הגדרה: **סגמנט** הינו אוסף שדות הקשורים בקשר לוגי ודרושים כיחידה אחת למטרות עיבוד הנתונים. הסגמנט הינו חלק של הרשומה הלוגית.



תרשים 3-15: חלוקה היררכית של הרשומה

ניתן להגדיר את הרשומה כאוסף של סגמנטים בעלי קשר לוגי ביניהם. הסגמנט מזכיר את מבנה הנגררות. רוב המערכות לניהול קבצים יודעות לטפל רק בסוג אחד של נגררות ולכן פותחו מערכות תוכנה אשר יודעות לטפל במספר כלשהו של סוגי נגררות עם מספר רב של מופעים.

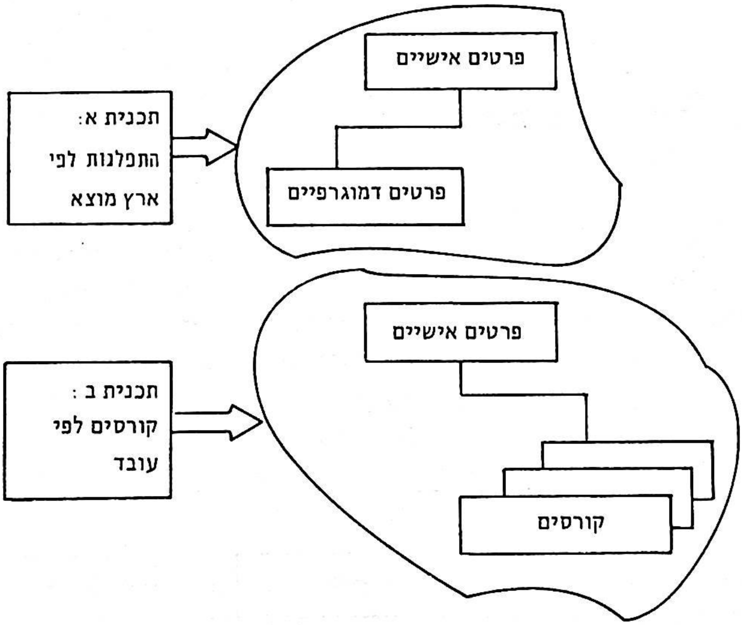


תרשים 3-16: מבנה היררכי של רשומת עובד

הפרטים המזהים ברשומת העובד שבתרשים 3-23 נמצאים **בסגמנט השורש** והם מזהים את כל הרשומה. כל שאר הסגמנטים מופיעים מבחינה היררכית מתחת לסגמנט השורש. סגמנט הפרטים הדמוגרפיים מופיע פעם אחת בלבד. סגמנט ההשכלה מופיע מספר פעמים, סגמנט אחד עבור כל רמת השכלה. הסגמנט של הקורסים יופיע גם כן מספר פעמים בהתאם למספר הקורסים שהעובד השתתף בהם.

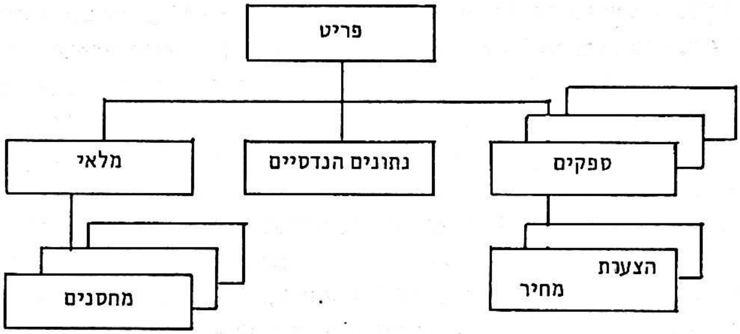
ייצוג היררכי זה הינו ייצוג טבעי יותר של הנתונים והוא עונה על שתי הבעיות שהועלו קודם. למרות שמבחינה לוגית האורך הכולל של הרשומה זהה בשתי השיטות הרי שמבחינה פיסית הנתונים לא חייבים להופיע זה ליד זה. ניתן לפזר את הסגמנטים השונים ביחידת האיחסון ובלבד שנבנה מנגנון שיקשר את כל הסגמנטים הדרושים יחדיו למבנה של רשומה. כל סגמנט כולל את השדות הדרושים בלבד, ולכן כל הסגמנטים מאותו סוג הם בעלי אורך שווה.

הסגמנטציה מאפשרת להציג לכל תוכנית רק את אותם חלקים (סגמנטים) מתוך הרשומה שדרושים לה. שתי תוכניות שונות יכולות "לראות" מבנים שונים: האחת אשר מפיקה דוח התפלגות העובדים לפי ארץ מוצא והשניה המפיקה דוח על הקורסים בהם השתתף כל עובד.



תרשים 3-17: הצגת חלק מהסגמנטים לתוכניות יישום שונות

בתרשים 3-17 כל תוכנית רואה מבנה לוגי הדרוש לה ואינה מטפלת בנתונים שאינם רלוונטים עבורה. בעיה שניה של מספר מופעים מוגבל של אינפורמציה בתוך הרשומה, נפתרת בשיטה זו מאחר ואין צורך לאחסן את כל הסגמנטים בצורה רציפה. כל סגמנט חדש יכתב במקום הפנוי ויקושר בצורה כלשהי (שנראה בהמשך) לרשומה שהוא שייך לה.
נבחן דוגמא נוספת:



תרשים 3-18: מבנה הירַרכי של רשומת פריט

ברשומת פריט שבתרשים 3-18 מספר המופעים של הסגמנטים עשוי להשתנות מרשומה לרשומה ולעיתים אפילו בצורה קיצונית. קיימים פריטים אשר נמצאים ב־30 מחסנים ואחרים נמצאים במחסן אחד בלבד. מספר המופעים של הסגמנטים הוא פונקציה המשתנה עם הזמן. ברגע נתון הפריט יכול להמצא ב־20 מחסנים וזמן מה אחר כך רק ב־10 מחסנים. שיטת הייצוג ההירַרכית נותנת שלשה ממדים לרשומה לעומת מימד אחד בשיטה הקודמת.

רוחב — ניתן לחלק את הרשומה למספר רב של סגמנטים,

עומק — לסגמנט יכולים להיות סגמנטים אחרים (כלומר קיימים קשרים של אב־בן כאשר הבן יכול להיות "אב" לסגמנטים אחרים),

עובי — מספר המופעים של כל סגמנט הוא בלתי מוגבל.

הפיכת המודל הלוגי של ייצוג רשומה בסגמנטים למודל פיסי לייצוג הרשומה על יחידות האחסון יכולה להיעשות בכמה שיטות. מאחר והרשומה מפוצלת לחלקים, כאשר כל חלק עשוי להופיע מספר בלתי מוגבל של פעמים לא ניתן ברוב המקרים לכתוב את הרשומה ביחידת האחסון כרצף אחד של תווים.

קישור הסגמנטים הדרושים בעזרת **מצביעים** (Pointers) יידון בפרק הדן במבנה נתונים משורשרים. קיימות מערכות לניהול נתונים המשתמשות במודל ההירַרכי לייצוג הנתונים, לדוגמא DL/1 של יבמ.

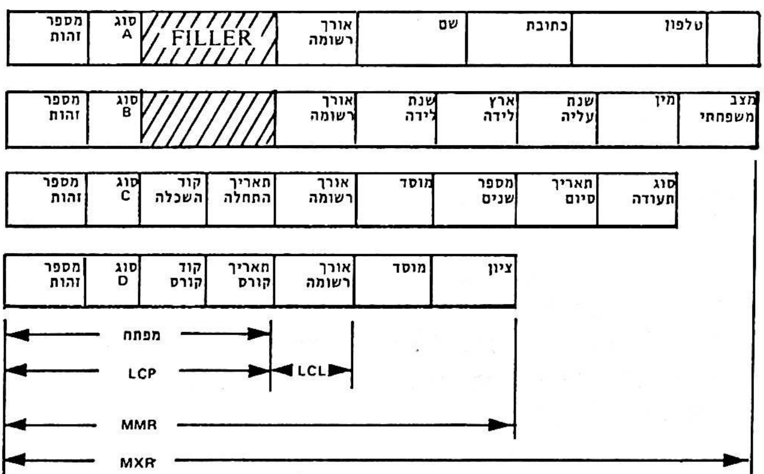
ניתן לקשר בין הסגמנטים הדרושים גם ללא מצביעים. במקרה זה יש לחזור על מפתח הרשומה בכל אחד מהסגמנטים. בשיטה זו מקבלים כפילות מסוימת

של נתונים על חשבון גמישות במבנה הנתונים. נדגים זאת בעזרת הדוגמא של קובץ עובדים. נשתמש בקובץ המאפשר רשומות באורך משתנה. נקבע תחילה את המפתחות וסוג הרשומה של כל אחד מהסגמנטים:

סגמנט פרטים אישיים — A — מספר זהות;
סגמנט פרטים דמוגרפיים — B — מספר זהות;
סגמנט השכלה — C — מספר זהות, קוד השכלה, תאריך;
סגמנט קורסים — D — מספר זהות, קוד קורס, תאריך קורס.

מאחר וסגמנטים C ו־D עשויים להופיע מספר רב של פעמים בקובץ יש להוסיף למפתח הרשומה שדות נוספים אשר יזהו בצורה חד־משמעית כל אחד מהמופיעים. בסגמנט הקורס אפשר להוסיף את קוד הקורס ותאריך תחילת הקורס כדי לאפשר זיהוי חד־משמעי של סגמנטים השייכים לאותו קורס אשר התקיים בתאריכים שונים או סגמנטים של קורסים שונים שהתקיימו באותו תאריך. בדוגמא שלפנינו הקשר בין סגמנט השורש (פרטים אישיים) לבין כל שאר הסגמנטים מבוצע באמצעות מספר הזהות. נתון זה יחזור על עצמו בכל הסגמנטים השייכים לאותה רשומה.

מאחר וניתן להשתמש בקובץ אחד ברשומות באורך משתנה, כל הרשומות יכולות להופיע פיסית באותו הקובץ כאשר כל אחת מהן באורך המיוחד לה. לכל אחת מהן יש להוסיף שדה אחד נוסף — אורך הרשומה. בהמשך נציג את הסגמנטים כרשומות פיסיות ונשתמש במונח "רשומה" לציון יחידת נתונים בעלת קשר לוגי מבלי להתחשב בתפקידים או במשמעותם בתוך הרשומה הלוגית של הפרט באוכלוסיה.



תרשים 3-19: דוגמא לסגמנטציה בעזרת רשומות באורך משתנה

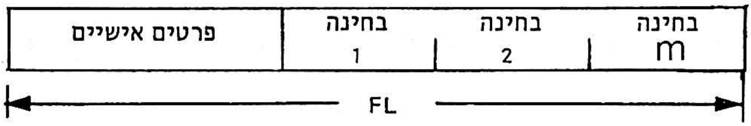
שימוש במפתחות חלקיים ומפתחות משניים יאפשר שליפה של כל רשומה לפי כל חתך רצוי. דוגמא מפורטת לכך תובא בפרק הדן בארגון נתונים עם אינדקסים משניים.

3.5 אירוע — תכנון רשומות

מטרת אירוע זה להציג את האלטרנטיבות השונות העומדות בפני מתכנן הרשומות תוך הדגשת היתרונות והחסרונות של כל אחת מהן.

משימה: יש לתכנן את מבנה הרשומה של מערכת לניהול ציוני בחינה של סטודנטים. על המערכת לנהל מספר פרטים אישיים לכל סטודנט ואת רשימת כל הקורסים בהם נבחן והציונים שהושגו בקורסים אלה.

אפשרות א׳: נחשב מראש את המספר המקסימלי של בחינות שסטודנט עשוי לגשת בתקופת לימודיו ונשתמש ברשומה באורך קבוע בהתאם לכך.



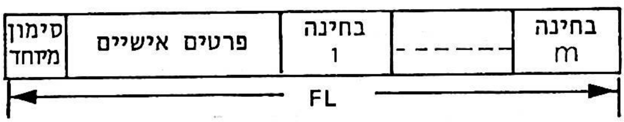
תרשים 3-20: רשומה באורך קבוע עם מספר מקסימלי של בחינות

יתרונות: — רשומות באורך קבוע.

חסרונות: — בזבוז גדול של שטח האחסון מאחר ויש לשער שאצל רוב הסטודנטים נמצא פחות בחינות מהמקסימום האפשרי.

— רשומה ארוכה מאד. המערכת לניהול קבצים תצטרך לבנות מאגרים גדולים בזכרון ולהשתמש בשטחי אחסון גדולים ביחידות האחסון החיצוניות.

אפשרות ב׳: נחשב את המספר הממוצע של בחינות שסטודנט יכול לגשת ונשאיר מקום בהתאם. לסטודנט הניגש למספר רב יותר של בחינות תיפתח רשומת־המשך עם סימון מיוחד.



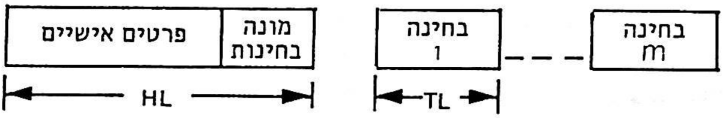
תרשים 3-21: רשומה באורך קבוע עם מספר ממוצע של בחינות

יתרונות: — רשומות באורך קבוע. הרשומות קצרות יותר מאשר באפשרות א׳.

חסרונות: — בזבוז בשטח האחסון מאחר ויהיו סטודנטים עם פחות בחינות מהממוצע. עם זאת, הבזבוז יהיה קטן יותר מאשר באפשרות א׳.

— תוכניות היישום מורכבות יותר. התוכניות צריכות לטפל ברשומות המשך על כל הכרוך בכך.

אפשרות ג': נשתמש ברשומות באורך משתנה. לכל בחינה שהסטודנט ניגש נפתח נגררת חדשה. המספר המקסימלי של הנגררות יהיה מספר הבחינות הממוצע (ניתן גם לבנות רשומות עם נגררות כמספר הבחינות המקסימלי אבל הן תהינה ארוכות מאד). אם יש סטודנט עם יותר בחינות מהממוצע תפתח לו רשומת המשך.



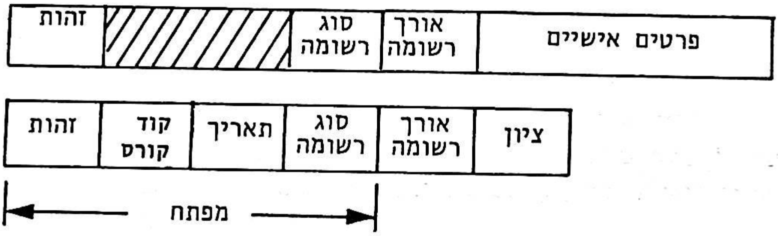
תרשים 3-22: רשומה של נגררות

יתרונות: — ניצול טוב יותר של שטח האחסון.

חסרונות: — שימוש ברשומות באורך משתנה.

— סיבוך תוכניות היישום בגלל הצורך לטפל ברשומות המשך.

אפשרות ד': יבנו שני סוגי רשומות. סוג אחד לניהול הפרטים האישיים וסוג שני לניהול הבחינות. כדי לקבל מפתח חד־משמעי יכלול המפתח של כל רשומה גם את תאריך הבחינה וקוד הקורס.



תרשים 3-23: שני סוגי רשומות בקובץ אחד

יתרונות: — אין מגבלה למספר הבחינות שסטודנט יכול לגשת.

— ניצול יעיל יחסית של שטח האחסון.

— רשומות קצרות יחסית ולכן דרושים גם מאגרים קטנים.

חסרונות: — כפילות מסוימת של נתונים. שדה תעודת זהות חוזר על עצמו בשני סוגי הרשומות.

— מספר גישות רב יחסית לקבלת כל הבחינות של סטודנט מאחר והן מפוזרות ברשומות שונות.

— סיבוך מסוים בתוכניות היישום. התוכניות צריכות לטפל בשני סוגי רשומות. לביטול בחינות סטודנט למשל, התוכנית צריכה לבצע חיפוש אחר כל רשומות הסטודנט.

קיימות אפשרויות נוספות המשלבות כמה מהאפשרויות שהוצגו ולמעשה אין כל כלל ומגבלה בענין זה. מנתח המערכות צריך להחליט על השיטה הטובה ביותר עבור היישום שהוא מתכנן, לשקול את היתרונות והחסרונות של השיטה בה בחר, ולבחון את האפשרויות של מערכת התוכנה לשרת בחירה זו.

הנחייה כללית לבחירת שיטת ארגון הרשומות היא להשתדל לבחור את השיטה הנוחה ביותר מבחינה לוגית ולא לנסות לנצל בצורה יעילה במיוחד את משאבי המחשב. זמן האחזקה הדרוש ליישום מתוחכם במיוחד יבטל כליל את כל היתרונות של ניצול טוב יותר לכאורה של משאבי המחשב.

3.6 שאלות

1. מדוע יש צורך ברשומות באורך משתנה?
2. הסבר כיצד רשומות ושדות באורך משתנה משפיעים על העיבוד.
3. קובץ המכיל 2000 רשומות באורך מקסימלי של 300 תווים, מעובד פעם בחודש. האם תבנה רשומות באורך קבוע או משתנה? האם תשובתך תשתנה אילו הקובץ הכיל 20,000 רשומות באורך מקסימלי של 1000 תווים ואורך מינימלי 100 תווים?
4. מהן לדעתך התכונות הנדרשות ממפתח הקובץ? מהם השיקולים לקביעת שדה המפתח?
5. עליך לתכנן מערכת מידע המנהלת את רישום הביקורים של חברי קופת־חולים במרפאות. תכנן את הרשומות ואת המפתחות שיש לנהל לדעתך. ציין מספר אלטרנטיבות והצג את היתרונות והחסרונות של כל שיטה.

פרק רביעי
שיטת ארגון סדרתי
(Sequential Access Method)

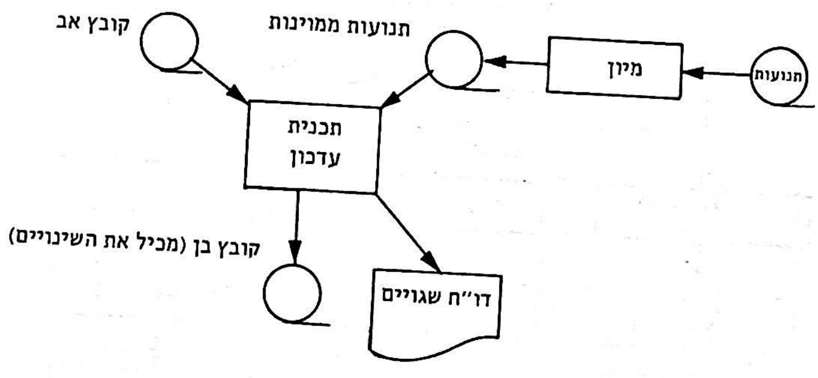
בפרק זה נציג את שיטת הארגון הבסיסית והפשוטה מבין כל שיטות ארגון הנתונים.

4.1 מבוא

שיטת הארגון הסדרתי היא שיטת ארגון הנתונים הבסיסית והפשוטה ביותר. בשיטה זו מאורגנות הרשומות בסדר פיסי הזהה לארגונן הלוגי. הרשומות מסודרות בקובץ אחת אחרי השניה בהתאם לסדר רישומן בקובץ. על מנת לסדר את הרשומות בסדר לוגי שונה כלשהו יש למיין את הרשומות תחילה בהתאם לסדר הדרוש. הקבצים הסדרתיים ניתנים ליישום בכל אמצעי האחסון הקיימים (סרטים מגנטים ודיסקים) להבדיל מהשיטות שיוצגו בהמשך וניתנות ליישום בדיסקים בלבד.

בזמן עיבוד הקובץ הסדרתי, הגישה לרשומות מתבצעת לפי סדר הופעתן בקובץ. אין אפשרות לדלג לפנים או לאחור וכל רשומה מעובדת בתורה. אין אפשרות להוסיף רשומות חדשות, לבטל רשומות קיימות או לעדכן רשומות קיימות. לביצוע פעולות העדכון יש להעתיק את כל הקובץ (קובץ האב) לקובץ חדש (קובץ בן) אשר יכיל את כל השינויים. כדי לבצע את העדכון, יש למיין את קובץ התנועות לפי הסדר שבו ממוין קובץ האב. העדכון מתבצע בזכרון המחשב והרשומה המעודכנת מועתקת לקובץ הבן.

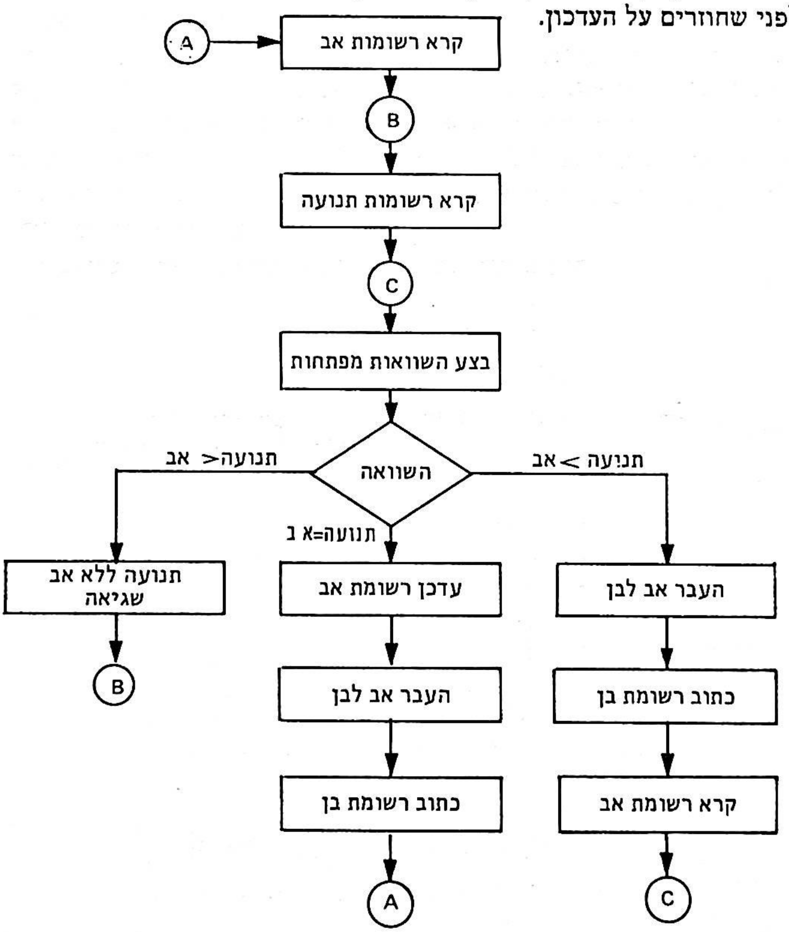
השימוש בקבצים בארגון סדרתי קיים במערכות אצווה.



תרשים 4-1: מהלך עדכון קובץ סדרתי

4.2 עדכון קובץ סדרתי

על מנת לעדכן קובץ סדרתי וממוין לפי מפתח כלשהו, יש למיין תחילה את התנועות לפי אותו מפתח על מנת שנוכל לבצע את העדכון של הקובץ במעבר אחד של התנועות. מאחר והעדכון לא משנה את רשומת האב, אין צורך במקרה של תקלה לשחזר את קובץ האב. יש להריץ פעם נוספת את התוכנית ולבנות קובץ־בן חדש. כפי שנראה בהמשך, בשיטות הארגון האחרות העדכון משנה את קובץ האב ולכן במקרה של תקלה יש לשחזר את הקובץ תחילה, לפני שחוזרים על העדכון.



תרשים 4-2: תרשים זרימה לעדכון קובץ סדרתי

נציג בצורה גרפית את תהליך העדכון של קובץ סדרתי. בקובץ התנועות מופיעים כל סוגי העדכון כמו פתיחת רשומות חדשות, שינוי ברשומות קיימות וביטול רשומות קיימות. בדוגמא נשתמש בקובץ מלאי וקובץ תנועות מלאי. כאשר המספר הקטלוגי ברשומה בקובץ האב שווה למספר הקטלוגי ברשומה בקובץ התנועות יש להוסיף את הכמות שברשומת התנועה לכמות שברשומת קובץ האב.

קובץ תנועות

סוג	כמות	מספר קטלוגי
פתיחה	10	10
פתיחה	15	12
שינוי	10	15
שינוי	10	17
שינוי	10	18
ביטול		30

קובץ אב

כמות	מספר קטלוגי
5	15
6	17
10	20
10	30
15	32

תוכנית עדכון

תנועות שגויות

שינוי	10	18

קובץ בן

10	10
15	12
25	15
16	17
15	32

* אי אפשר לעדכן פרטים ברשומה שאינה קימת בקובץ אב

תרשים 4-3: דוגמא לעדכון קובץ סדרתי בשיטת אב־בן

שיטת העבודה בקובץ הסדרתי היא כזאת שאין כמעט חשיבות למספר רשומות התנועה מאחר ובכל מקרה מטפלים בכל רשומות האב, גם אם אין להן רשומות מקבילות בקובץ התנועות. חוסר רגישות השיטה למספר רשומות התנועה היא ההופכת את השיטה ליעילה כאשר יש לעבד את כל הקובץ (ביישום של משכורת, למשל) או את רובו (כאשר יש עדכון לרוב הרשומות).

הגדרת קובץ סדרתי בקובול

קובץ סדרתי מוגדר בקובול במשפט SELECT.

```
SELECT MLAY-FILE ASSIGN TO XX ZZ
       ORGANIZATION IS SEQUENTIAL.
===
FD      MLAY-FILE
        DATA RECORD IS MLAY-REC.
01      MLAY-REC.
       03 MIS-QATALOGI            PIC 9(9).
       03 MAHSAN                  PIC 9(3).
       03 KAMUT                   PIC 9(8)V99.
```

העיבוד של הקובץ מתבצע בעזרת פקודות READ של קובץ האב ופקודות WRITE לקובץ הבן.

```
PROCEDURE DIVISION
= = =
OPEN   INPUT MLAY-FILE
       OUTPUT MLAY-OUT.
= = =
READ   MLAY-FILE AT END
       PERFORM SOF.
PERFORM TIPUL.
WRITE MLAY-OUT FROM MLAY-REC.
= = =
CLOSE  MLAY-FILE MLAY-OUT.
= = =
```

4.3 שיטות חיפוש בקובץ סדרתי

תהליכי עיבוד הנתונים דורשים לעיתים איתור רשומה בקובץ לשם עדכון או לשם עיבוד אחר. תהליך זה יבוצע בקובץ סדרתי כאשר מספר הרשומות המבוקשות קטן מאד בהשוואה לגודל הקובץ.

שיטת החיפוש הפשוטה ביותר תהיה על ידי סריקת הקובץ מתחילתו, עד לאיתור ברשומה המבוקשת (לפי מפתח או לפי זיהוי אחר). הסריקה נעשית על ידי השוואה של מפתח כל רשומה ורשומה בקובץ אל המפתח המבוקש. אם הקובץ מכיל N רשומות, יש לבצע בממוצע N/2 השוואות.

הזמן הדרוש לקריאת רשומה הוא R/t כלומר גודל הרשומה מחולק במהירות העברת הנתונים. זמן הסריקה הממוצע יהיה:

$$\overline{T} = \frac{1}{2}N\frac{R}{t}$$

זמן הסריקה המקסימלי יהיה:

$$T_{max} = N\frac{R}{t}$$

חיפוש לפי גושים (Block Search)

כאשר הרשומות ממוינות לפי מפתח אין צורך לסרוק את כל הרשומות בגוש. ניתן לבצע השוואה של מפתח הרשומה האחרונה בגוש למפתח הרשומה הדרושה. אם המפתח הדרוש גדול יותר נקרא גוש נוסף ונחזור על ההשוואה. אם המפתח הדרוש קטן יותר, יש לסרוק את כל הגוש לאיתור הרשומה המבוקשת.

חיפוש בינרי (Binary Search)

כאשר הקובץ ממוין ונמצא על דיסק, קיימת אפשרות להשתמש בשיטת החיפוש הבינרי לאיתור הרשומה. שיטה זו מצמצמת בצורה ניכרת את מספר הגישות שיש לבצע וכתוצאה — את זמן איתור הרשומה.

החיפוש הבינרי הינו שיטת החיפוש המהירה ביותר לאיתור רשומות בקובץ סדרתי. תהליך החיפוש הבינרי הינו רצף של השוואות של מפתח רשומות למפתח המבוקש. ההשוואה הראשונה בשיטה זו מתבצעת באמצע הקובץ. אם המפתח של הרשומה האמצעית גדול מהמפתח הדרוש תתבצע ההשוואה הבאה במחצית הראשונה של הקובץ. אם המפתח של הרשומה האמצעית נמוך מהמפתח המבוקש תבוצע ההשוואה במחצית השנייה של הקובץ. בהתאם להשוואה מחליטים איזה חצי מהקובץ לחצות שוב ולהתקדם להשוואה הבאה. מאחר וכל פעם חוצים לשניים את מספר הרשומות שנותרו, מספר ההשוואות a שנצטרך לבצע בשיטה זו יהיה: $2^a = n \Rightarrow a = \log_2 n$

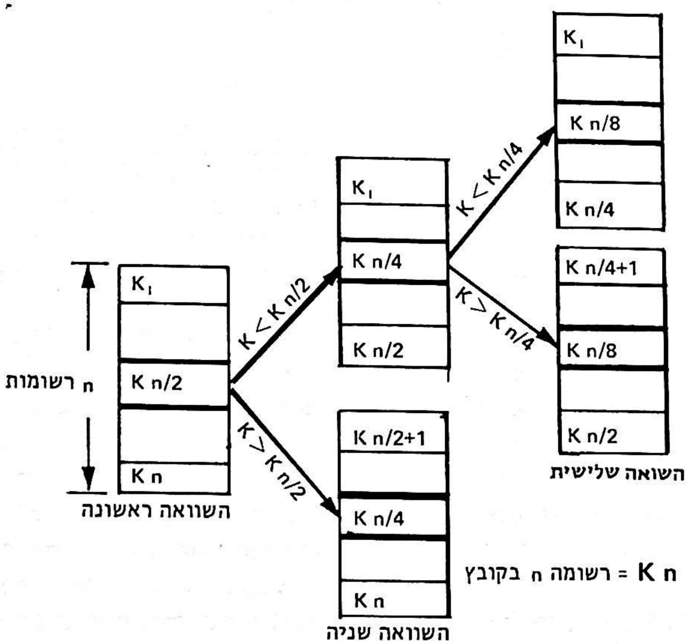
כלומר, מספר ההשוואות הדרוש יהיה יחסי ל-$\log_2$ של מספר הרשומות בקובץ. אם n אינו חזקה מלאה של 2 יש לבצע $[\log_2 n] + 1$ השוואות כאשר $[\log_2 a]$ היא המספר השלם הגדול העוקב.

לדוגמא: אם הקובץ מכיל 10,000 רשומות יש לבצע 14 השוואות $(\log_2 10{,}000 + 1)$ לאיתור הרשומה לעומת 5,000 השוואות אשר דרושות בחיפוש ליניארי רגיל.

מאחר והקובץ עצמו מאוחסן בגושים ניתן ליעל את השיטה. במקום לחצות לשנים את מספר הרשומות נחצה לשניים את מספר הגושים. בהתאם למפתח הראשון והאחרון בגוש נחליט לאיזה כיוון להמשיך את החיפוש. בצורה זו מספר הגישות לדיסק יהיה יחסי ל-$\log_2$ של מספר הגושים ולא של מספר הרשומות. עבור קבצים גדולים זהו הבדל משמעותי ביותר.

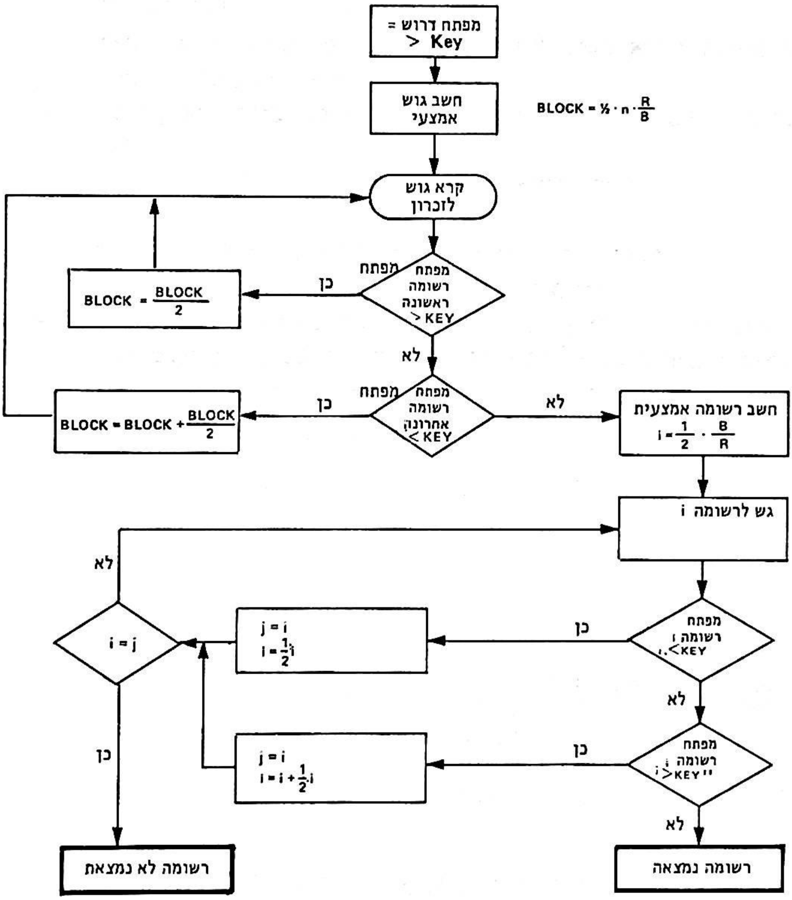
מאחר ובכל גוש יש B/R רשומות הרי שבכל הקובץ יש nR/B גושים. זמן הגישה לגוש אחד מורכב מזמן החיפוש S, זמן ההשהיה הסיבובית r, וזמן העברת הגוש אל הזכרון B/t. זמן החיפוש הכולל בקובץ סדרתי בשיטת החיפוש הבינרי יהיה:

$$T = \log_2 \left(n\frac{R}{B}\right) \cdot \left(S + r + \frac{B}{t}\right)$$



תרשים 4-4: השוואות בחיפוש בינרי

נציג כאן את תרשים הזרימה לביצוע חיפוש בינרי לפי גושים:



תרשים 4-5: תרשים זרימה לחיפוש בינרי

4.4 סיכום

לשיטת הארגון הסדרתי יתרונות וחסרונות.

יתרונות

— מבנה נתונים פשוט ביותר. איתור הרשומות מתבצע על־ידי קריאת הרשומה הבאה.
— כאשר שיעור הפעילות גבוה, הפשטות של שיטת איתור הרשומות הופך את השיטה לכדאית.
— העדכון אינו הורס את תוכן רשומות האב. קל להתאושש מתקלות בארגון זה.
— ניתן להשתמש גם בסרטים וגם בדיסקים לאיחסון הקובץ.

חסרונות

— יש לעבד תמיד את כל הקובץ ללא קשר לשיעור הפעילות.
— יש למיין תחילה את רשומות התנועות לפני העדכון.
— עדכון הקובץ מסתיים רק לאחר עדכון הרשומה האחרונה כאשר פועלים בסרטים מגנטיים. עדכון בדיסק אפשר להפסיק ולהמשיך מחדש.
— אין אפשרות לגשת לרשומה באופן ישיר (בסרטים דרוש חיפוש עוקב ובדיסקים אפשרי חיפוש בינרי).

פרק חמישי
ארגון אינדקס סדרתי
(Index Sequential Access Method)

בפרק זה נציג שיטה לארגון נתונים המשלבת את התכונות הסדרתיות של הקובץ וגישה ישירה.

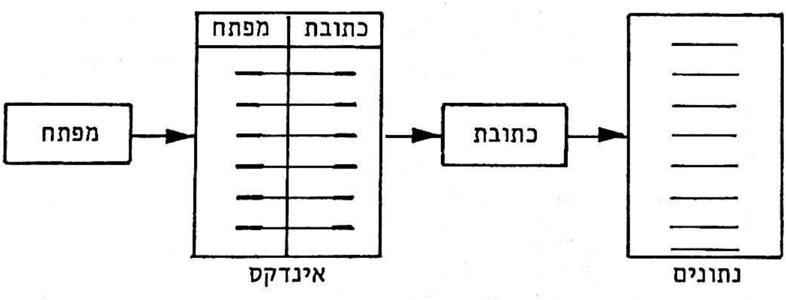
5.1 מבוא

בעית איחזור המידע הינה אחת הבעיות המרכזיות במערכות מידע ממוחשבות. ארגון הקבצים הסדרתי מאפשר איחזור רשומות רק לפי הסדר הפיסי שלהן בקובץ ולא לפי מפתח לוגי של הרשומה. פיתוח יחידות איחסון המאפשרות גישה ישירה הביא לפיתוח מקביל של שיטות ארגון נתונים המסתמכות על הגישה הישירה ומאפשרות איחזור רשומות באופן יעיל. כל השיטות לאיחזור נתונים על סמך מפתח לוגי מופעלות רק ביחידות איחסון אלה.

קיימות מספר שיטות לאיחזור רשומות לפי מפתח. בפרק זה נסקור את השיטות השונות המשתמשות באינדקסים.

5.2 איחזור רשומה באמצעות אינדקס

האינדקס הינו טבלה בעזרתה מתבצע תהליך תרגום בין המפתח של הרשומה לבין כתובתה ביחידת האחסון. האינדקס הוא אמצעי המאפשר גישה מהירה לרשומה בקובץ מבלי שיהיה צורך בסריקת כל הקובץ. המפתח של הרשומה יכול להיות שדה כלשהו ברשומה המזהה את הרשומה ומאפשר את איחזורה. האינדקס ממוין לפי המפתח.



תרשים 5-1: תרגום מפתח לכתובת בעזרת אינדקס

דוגמא לאיחזור רשומת מלאי לפי מפתח:

```
MOVE     "12845" TO MISP-CATALOGI.
READ     MLAY-FILE INVALID KEY
         DISPLAY "RECORD NOT FOUND"
         GO TO NOT-FOUND.
```

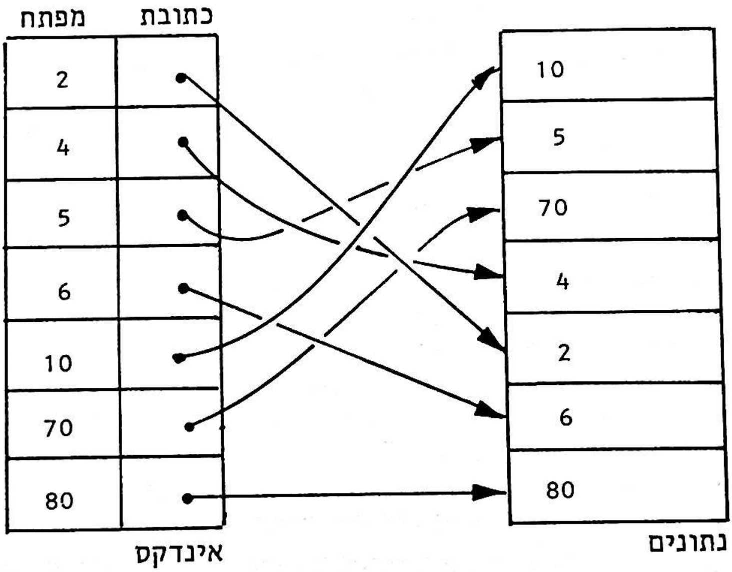
התוכנית מעבירה את תוכן המפתח של הרשומה המבוקשת לשדה המפתח. המערכת לניהול קבצים מבצעת את סריקת האינדקסים לאיתור כתובת הרשומה ולאחר מכן מאתרת את הרשומה בקובץ. כאשר פעולת הקריאה (READ) אינה מצליחה לאתר את הרשומה (INVALID KEY) היא מודיעה על כך.

קיימים שני סוגי אינדקסים: אינדקס ממצה ואינדקס סלקטיבי.

אינדקס ממצה (Exhaustive Index)

אינדקס ממצה מכיל רשומת אינדקס אחת עבור כל רשומת נתונים. לדוגמא נשתמש בקובץ נתונים הנמצא בדיסק ואיננו ממוין לפי מפתח. על מנת לגשת לרשומות בקובץ בצורה מהירה נבנה אינדקס הממוין לפי השדה הדרוש. רשומת האינדקס תכיל את ערך המפתח ואת כתובת הרשומה בדיסק. לאיתור רשומה נסרוק את האינדקס (מאחר והוא ממוין נוכל לסרוק אותו בשיטה בינרית). רשומת האינדקס המכילה את המפתח המבוקש מכילה גם את הכתובת הפיסית של הרשומה בדיסק. קריאת הרשומה תבוצע בגישה ישירה.

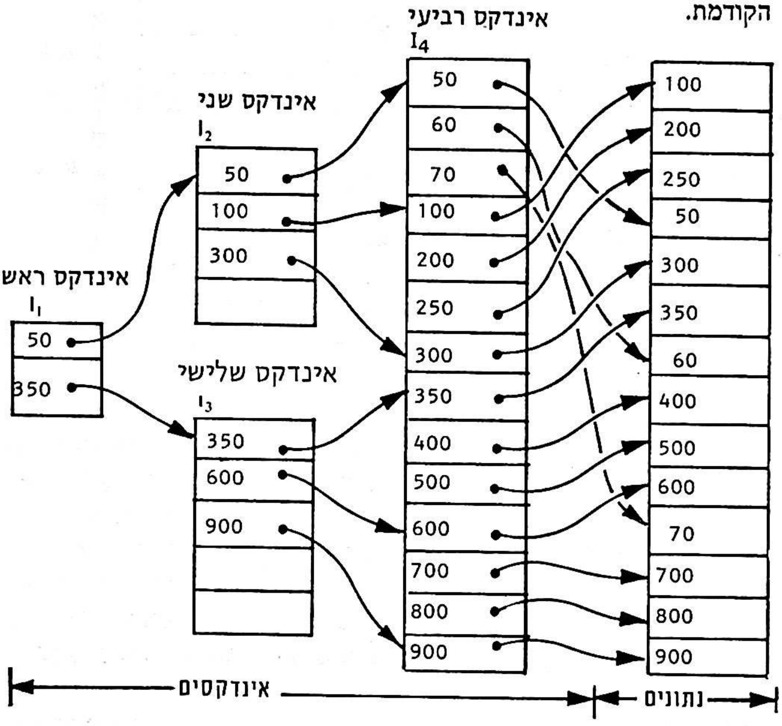
מקובל להשתמש באינדקס ממצה כאשר המפתח הדרוש אינו מפתח הסדר. נראה שימוש בסוג זה של אינדקסים בפרק הדן באינדקסים משניים.



תרשים 5-2: אינדקס ממצה

כאשר מספר הרשומות בקובץ הוא גדול, גם מספר רשומות האינדקס יהיה גדול. על מנת ליעל את תהליך החיפוש ניתן לבנות מספר רמות נוספות של אינדקס.

תהליך החיפוש אחר רשומה יבוצע בצורה הבאה: בכל אחת מרמות האינדקס יש לסרוק את הרשומות עד לאיתור מפתח הגדול מהמפתח הדרוש. יש לגשת לרמה הבאה באינדקס בהתאם למצביע של רשומת האינדקס הקודמת.



תרשים 5-3: אינדקס ממצה עם מספר רמות

לדוגמא נחפש רשומה עם מפתח 700. תהליך החיפוש יהיה:

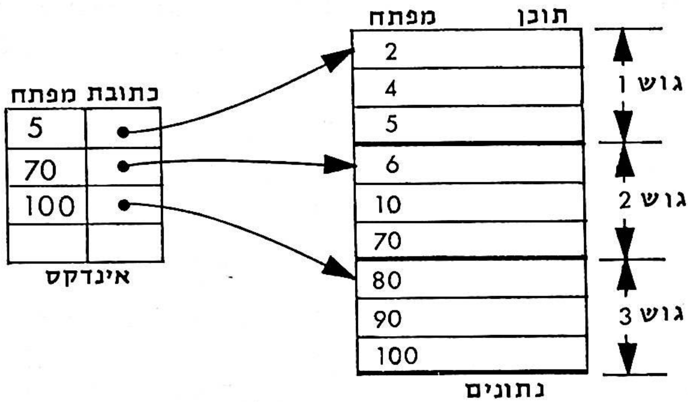
— חיפוש ברמת אינדקס I_1. האינדקס הזה מצביע על אינדקס I_3.
— חיפוש באינדקס I_3. המפתח 900 גדול מ־700 ולכן נשתמש במצביע המופיע ברשומה הקודמת המפנה אותנו לקטע מתוך האינדקס I_4.
— חיפוש סדרתי באינדקס I_4 החל מרשומת אינדקס עם מפתח 600. לאחר השוואה אחת מגיעים לרשומה עם מפתח 700. ברשומה זו יש מצביע לרשומת הנתונים.

השיפור בזמן החיפוש נובע מכך שבשיטה זו דרושות 6 השוואות לעומת 12 הדרושות לאיתור הרשומה עם אינדקס ממצה ברמה אחת בלבד. נציין שהשיפור הוא על חשבון מקום איחסון נוסף הדרוש לאינדקסים הנוספים וצריכת יע"מ גבוהה יותר לניהולם.

אינדקס סלקטיבי (Selective Index)

אינדקס סלקטיבי מכיל רשומת אינדקס אחת עבור קבוצת רשומות נתונים. בדרך כלל האינדקס מכיל רשומת אינדקס אחת עבור כל גוש. בשיטה זאת יש

למיין תחילה את הרשומות בקובץ. האינדקס מכיל את המפתח של הרשומה בעלת הערך הגבוה ביותר (או הנמוך ביותר) הדרוש לאיתור הרשומה בגוש.



תרשים 5-4: מבנה אינדקס סלקטיבי

איתור רשומה בקובץ יתבצע בצורה הבאה:

— קריאת רשומות אינדקס עד אשר המפתח באינדקס גדול או שווה למפתח הרשומה הדרושה.
— גישה לגוש אשר כתובתו מופיעה ברשומת האינדקס וקריאתו לזכרון.
— חיפוש סדרתי בתוך הגוש עד לאיתור הרשומה הדרושה. אם הרשומה איננה בגוש, יש להודיע לתוכנית שהרשומה אינה נמצאת בקובץ.

בדוגמא פשוטה זו יש לבצע שתי גישות לדיסק עבור כל פעולת איתור: אחת לקריאת האינדקס ואחת לקריאת גוש נתונים. החיפוש בתוך הגוש נעשה בזכרון ואינו כרוך בפעולת קריאה פיסית. מאחר והנתונים עצמם ממוינים לפי מפתח החיפוש נהוג לקרוא לשיטה זו **ארגון אינדקס סדרתי.**

ניתן להשתמש באינדקס הסלקטיבי גם בצורה נוספת. במקום לבנות רשומת אינדקס אחת עבור כל גוש של נתונים נבנה רשומת אינדקס אחת המכילה חלוקה כלשהי של הנתונים אשר אינה חייבת להיות החלוקה הפיסית של הנתונים (בגושים למשל) כפי שראינו קודם. אם המפתח מכיל שני חלקים למשל, אות וסיפרה, ואם קובץ הנתונים ממוין, נוכל לבנות אינדקס המכיל רשומה אחת עבור כל אות המתחילה את המפתח.

בתרשים 5-5 רשומת האינדקס מצביעה על הרשומה הראשונה המתחילה באות המתאימה. לדוגמא, רשומת האינדקס המכילה את האות B מצביעה על הרשומה הראשונה שהמפתח שלה מתחיל באות B. ברגע שמגיעים לרשומת הנתונים מתחיל תהליך של חיפוש אחר הרשומה הדרושה. לשיטה זו של בנית אינדקס מקובל לקרוא Digital Index מאחר ובונים את האינדקס ומטפלים במפתחות וחלקיהם בצורה עשרונית. אם לקובץ יש d חלוקות כאלה, מספר החיפושים m שיש לבצע יהיה בקירוב:

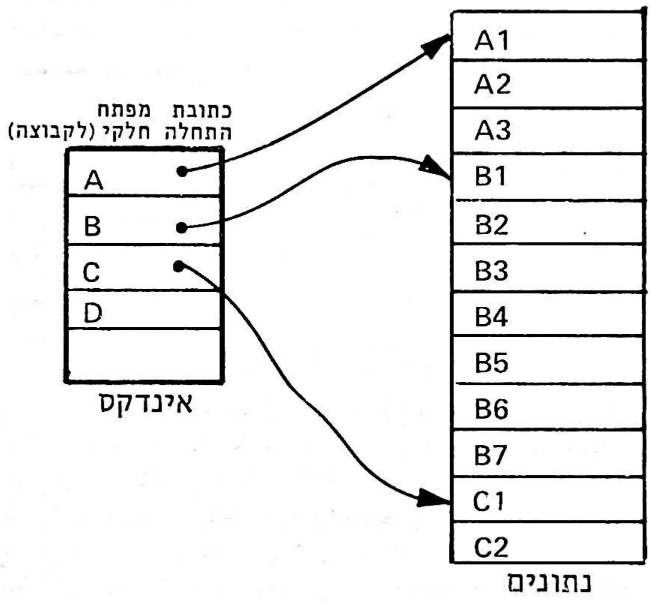
$$m = 1 + \frac{n}{d} \cdot \frac{1}{2}$$

מספר הגישות הכולל הוא סכום גישה אחת לאינדקס ועוד כמחצית הרשומות המתאימות לחלוקה מסוימת. לדוגמא, אם בקובץ יש 100,000 רשומות המחולקות ל־20 מחלקות שונות באופן אחיד מספר הגישות הממוצע לאיתור רשומה יהיה:

$$m = 1 + \frac{100.000}{20} = 5001$$

אם החיפוש של הרשומה היה מתבצע באופן סידרתי רגיל צריך היה לבצע 50,000 השוואות בממוצע.

דוגמא:



תרשים 5-5: אינדקס לפי האות הראשונה של המפתח

נחשב את גודל הגוש האופטימלי עבור רמת אינדקס אחת בלבד. אם הקובץ מכיל n רשומות וגודל הגוש B, מספר רשומות האינדקס יהיה $K = n/B$.

מספר החיפושים הכולל שיש לבצע בממוצע הוא כמחצית מספר רשומות האינדקס וכמחצית הרשומות בגוש הנתונים.

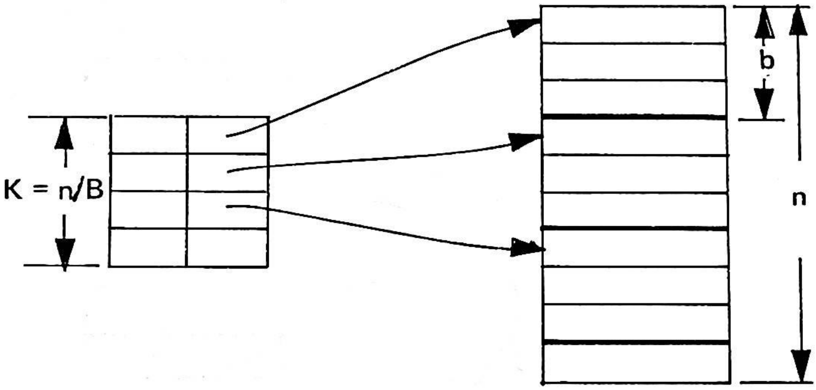
$$S = \frac{1}{2} \cdot \frac{n}{B} + \frac{B}{2}$$ (מספר חיפושים)

נגזור את הביטוי לשם מציאת מספר הרשומות בגוש הנתונים האופטימלי:

$$\frac{dS}{dB} = -\frac{n}{2B^2} + \frac{1}{2} = 0$$

$$B^2 = n$$

$$\boxed{B = \sqrt{n}}$$



תרשים 5-6: חישוב גודל גוש אופטימלי

המסקנה היא שמספר החיפושים יהיה מינימלי כאשר הגוש יכיל $\sqrt{n}$ רשומות, כלומר כאשר מספר הרשומות בגוש יהיה שורש רבועי של מספר הרשומות בקובץ הנתונים.

אם מספר רשומות האינדקס גדול כתוצאה ממספר רב של רשומות נתונים, ניתן לבנות מספר **רמות אינדקס** (Multiple Level Index).

לדוגמא נציג תהליך איתור רשומה בקובץ עם שתי רמות אינדקס. נניח שמחפשים רשומה עם מפתח לוגי 722:

— קרא גוש מאינדקס רמה עליונה לזכרון (גוש I_1).

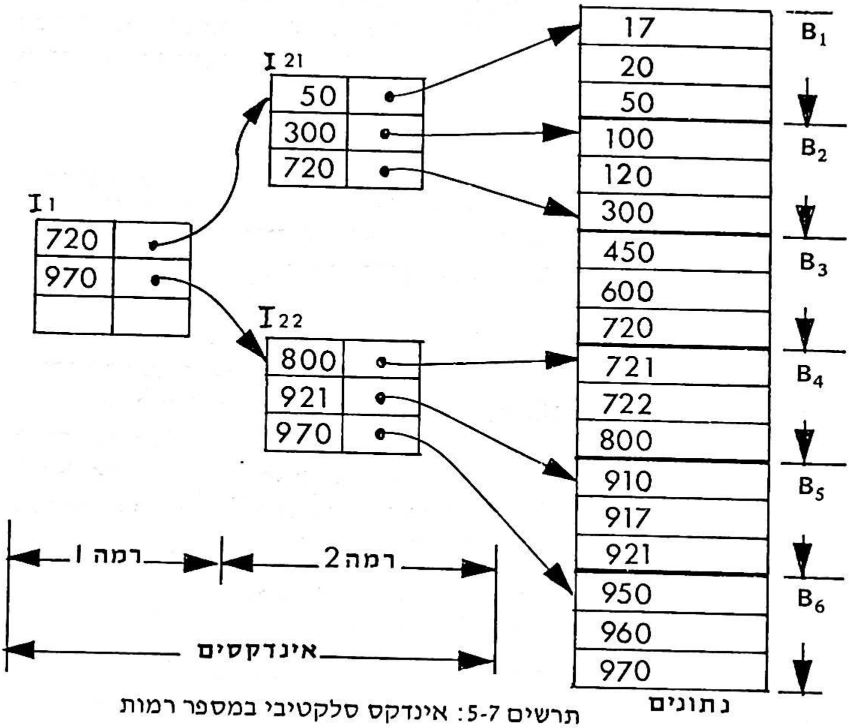
— חפש בגוש עד למציאת מפתח ראשון הגדול ממפתח החיפוש. התוצאה היא מפתח 970. רשומה זו מצביעה על גוש אינדקס I_{22}.

— קרא גוש אינדקס I_{22} לזכרון.

— חפש בגוש עד למציאת מפתח ראשון הגדול ממפתח החיפוש. התוצאה היא מפתח 800. רשומה זו מצביעה על גוש נתונים B_4.

— קרא גוש נתונים B_4 לזכרון.

— חפש בגוש עד למציאת רשומה עם המפתח הדרוש. אם נמצא — העבר את הרשומה לעיבוד. אם לא — יש להודיע לתוכנית שהרשומה לא נמצאה.



תרשים 7-5: אינדקס סלקטיבי במספר רמות

למספר רמות האינדקס חשיבות רבה בקביעת זמן הגישה לרשומה בקובץ אינדקס סדרתי. מספר הגישות הוא יחסי למספר רמות האינדקס. לדוגמא, עבור שלש רמות אינדקס יש לבצע שלש גישות לדיסק, שתי גישות לאינדקס וגישה אחת לגוש הנתונים. לשם יעול תהליך החיפוש מחזיקים בזכרון את האינדקס ברמה העליונה, מאחר והוא קטן יחסית. ההעברה לזכרון המחשב נעשית על ידי התוכנה לניהול קבצים לאחר פתיחת הקובץ. גודל הגוש האופטימלי, כאשר יש L רמות אינדקס וכאשר n הוא מספר הרשומות בקובץ:

$$B = \sqrt[L]{n}$$

בנוסף לפרק זה ניתנת השיטה לחישוב מספר רמות האינדקס כאשר נתון: מספר הרשומות בקובץ (n), גודל הגוש (B), גודל הרשומה (R), גודל המפתח (K) וגודל הכתובת של הגוש (P):

$$L \cong \frac{\log (n\frac{R}{B})}{\log (\frac{B}{P+K})}$$

ככל שגודל הגוש של האינדקס גדול יותר, מספר רמות האינדקס קטן.

נשתמש בדוגמא:

נתון קובץ עם 100 מיליון רשומות. גודל הגוש הוא 5000 תווים. אורך המפתח של הרשומה 44 תווים. אורך הכתובת 6 בתים. אורך הרשומה 500 תווים. מספר רמות האינדקס הדרושות:

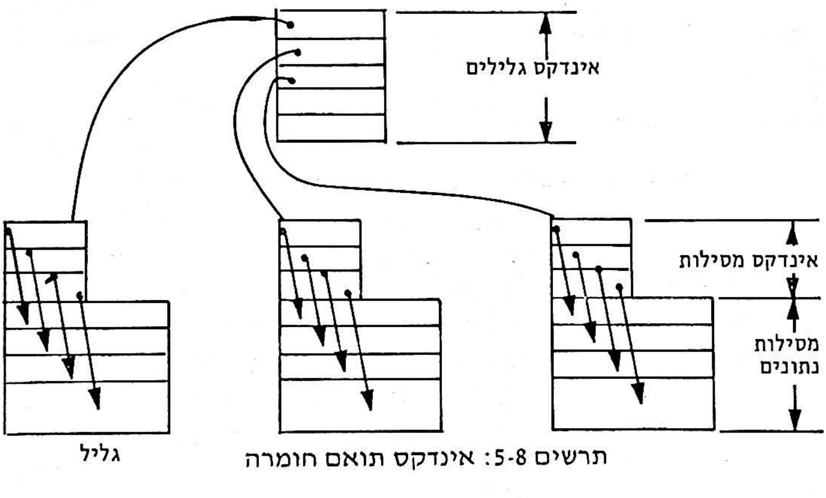
$$L \cong \frac{\log\left(10^8 \frac{500}{5000}\right)}{\log\left(\frac{5000}{500}\right)} = \frac{8}{2} = 4$$

אינדקס תואם חומרה (Hardware Oriented Index)

קיימת אפשרות לתכנן את האינדקס תוך התבססות על מבנה החומרה, כלומר מבנה הדיסק המכיל את הקובץ. לאינדקס זה יהיו שתי רמות — רמת הגליל ורמת המסילה.

— **אינדקס הגלילים** (Cylinder Index) אשר מכיל את המפתח הגבוה ביותר של רשומה הנמצאת באותו גליל, ואת הכתובת הפיסית של הגליל.

— **אינדקס המסילות** (Track Index) נמצא במסלול הראשון של כל גליל וכולל את אינדקס המסילות של אותו גליל בלבד. עבור כל מסילה השייכת לגליל תהיה "כניסה" באינדקס אשר תכיל את המפתח הגבוה ביותר של רשומה באותה מסילה ואת הכתובת הפיסית שלה.



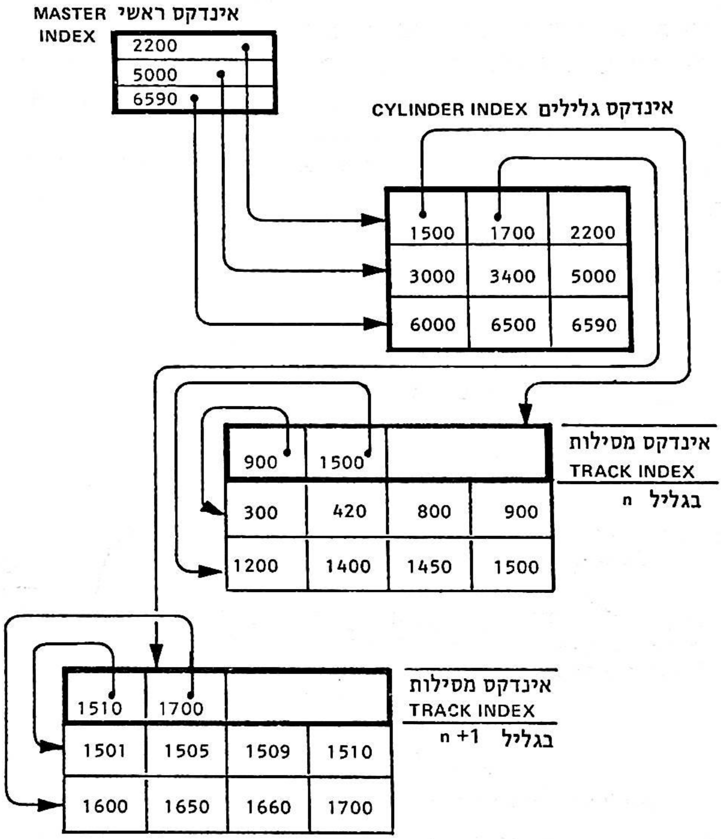
תרשים 5-8: אינדקס תואם חומרה

בדרך כלל אינדקס כזה נוטה להיות בעל פחות רמות אינדקס מאשר **אינדקס תואם תוכנה** (Software-Oriented Index). הבעיה היא שהשיטה פחות גמישה ותלויה בסוג הדיסק עליו מאוחסן הקובץ. העברת הקובץ לדיסק בעל מבנה אחר תגרום לשינוי האינדקסים השונים.

על מנת ליעל את החיפוש באינדקס הגלילים ניתן להוסיף רמה נוספת של אינדקס, **אינדקס ראשי** (Master Index) המכיל את המפתח הגבוה ביותר הנמצא בקבוצת גלילים. מאחר ואינדקס הגלילים עצמו יכול להיות גדול הוא ירשם במספר גושים פיסיים. האינדקס הראשי יכול במקרה זה להצביע על

הרשומה הגבוהה ביותר בכל גוש של אינדקס הגלילים כדי שאפשר יהיה לגשת באופן ישיר לגוש המכיל את הפרטים המבוקשים.

דוגמא לאינדקס תואם חומרה מופיע בתרשים 5-9.



תרשים 5-9: דוגמא לאינדקס תואם חומרה

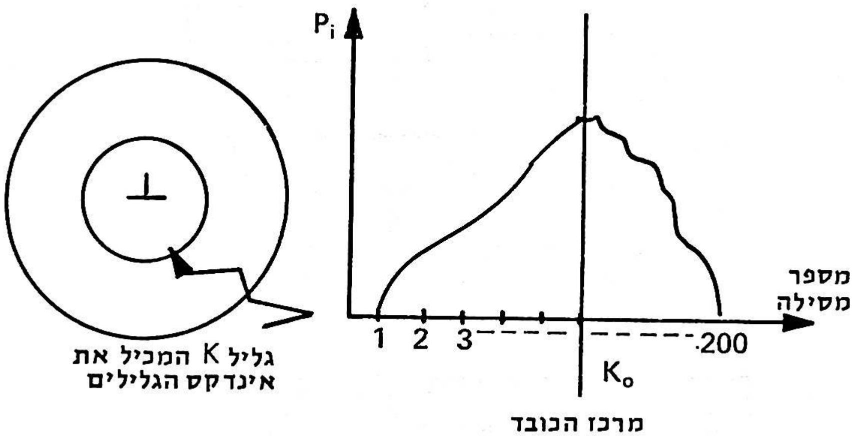
מיקום אינדקס הגלילים

נבחן עתה את המקום שרצוי לבחור בו לשמירת אינדקס הגלילים. בדרך כלל האינדקס הראשי מוחזק באופן קבוע בזכרון ולכן מיקומו על פני הדיסק חסר חשיבות מאחר וניגשים אליו רק פעם אחת. המיקום של אינדקס הגלילים הוא חשוב מאחר וכל גישה לקובץ עוברת דרכו. לאחר שמתבצע חיפוש בתוך אינדקס הגלילים יש להזיז את הזרוע לפנים או לאחור כדי להגיע לגליל הרצוי לשם חיפוש באינדקס המסילות.

אם יודעים את ההסתברות P_i לגשת לנתון הנמצא בגליל i המקום האופטימלי של אינדקס הגלילים יהיה בגליל של מרכז הכובד של ההתפלגות.

כלומר, השטח שמשמאל ל־K_0 שווה לשטח שמימין ל־K_0. אם נמקם את אינדקס הגלילים בגליל K_0, אורך הדרך שצריך יהיה לעשות בממוצע עד לגליל המתאים תהיה מינימלית.

בדרך כלל בתחילת השימוש בקובץ ההסתברויות אינן ידועות, ולכן מקובל למקם את האינדקס בגליל האמצעי של הקובץ. במשך פעילות הקובץ ניתן לעקוב אחר התפלגות הגישות ולשפר את הביצועים על־ידי הזזת מיקום האינדקס.



תרשים 5-10: מיקום אינדקס הגלילים בדיסק

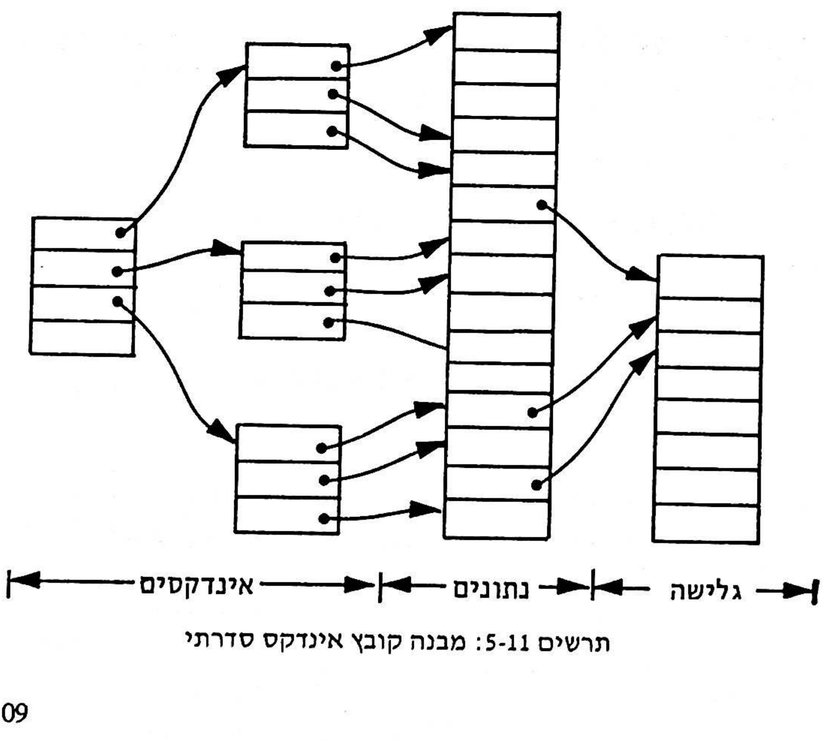
5.3 ארגון אינדקס סדרתי
(ISAM — Indexed Sequential Organization)

מטרת שיטת הארגון האינדקס סדרתי היא לאפשר גישה סדרתית לקובץ ביעילות גבוהה וגישה מהירה לחלק מהרשומות בעזרת אוסף אינדקסים. השיטה מאפשרת הוספה, עדכון, ביטול רשומות מבלי שיהיה צורך בהעתקת הקובץ כמו בשיטה הסדרתית. בשיטת ארגון זו מוחזקות הרשומות בסדר לוגי שאינו זהה בהכרח לסדר הפיסי שלהן בדיסק.

מבנה קובץ אינדקס סדרתי

הקובץ מורכב משלושה שטחים שונים:

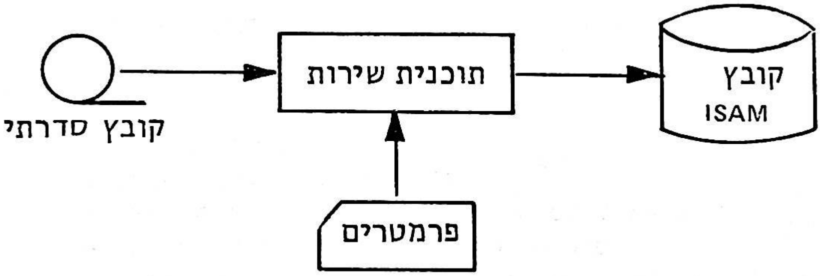
— איזור הנתונים שבו מאוחסנות הרשומות בצורה ממוינת,
— איזור אינדקסים שבו מאוחסנות רשומות האינדקס,
— איזור גלישה שבו מאוחסנות רשומות שאין להן מקום באיזור הנתונים אף־על־פי שהן שייכות לו בהתאם לסדר המיון שלהן.



תרשים 5-11: מבנה קובץ אינדקס סדרתי

איזור הנתונים מכיל את הרשומות בסדר עוקב בהתאם למפתח הרשומה. בזמן בניית הקובץ, נטענות כל הרשומות על ידי מערכת התוכנה לניהול קבצים בסדר עוקב והיא בונה גם את האינדקסים השונים תוך כדי תהליך הטעינה. לאחר בנית הקובץ, אין כל רשומה באיזור הגלישה. רשומות "גולשות" לאיזור זה במהלך עדכון הקובץ, כפי שנראה בהמשך.

לבנית הקובץ משתמשת המערכת לניהול קבצים במספר נתונים כגון: אורך הרשומה, אורך המפתח ומיקומו ברשומה, מבנה הרשומה (קבוע או משתנה), גודל הגוש ועוד. לרוב מערכות התוכנה יש תוכנית שירות (Utility) מיוחדת בעזרתה ניתן לבנות קובץ ISAM מתוך קובץ סדרתי.



תרשים 5-12: בנית קובץ ISAM בעזרת תוכנית שירות

בתוך תוכנית קובול משתמשים במשפט SELECT להגדרת קובץ אינדקס סדרתי ולהגדרת המפתח של הרשומה.

```
SELECT MLAY-FILE ASSIGN TO XX
    ORGANIZATION IS INDEXED
    RECORD KEY IS MISP-CATALOGI.

FD  MLAY-FILE
    LABEL RECORD IS OMITTED
    DATA RECORD IS MLAY-REC.

01  MLAY-REC.
    03  SUG                      PIC XX.
    03  MISP-CATALOGI            PIC 9(9).
    03  MAHSAN                   PIC 99.
    03  KAMUT                    PIC 9(8)V99.
```

שיטות פניה לקובץ אינדקס סדרתי

קיימות שלש שיטות פניה שונות לקובץ אינדקס סדרתי:

— סדרתית (Sequential)

— אקראית (Random)

— דינמית (Dynamic)

פניה סדרתית מאפשרת את קריאת הקובץ מתחילתו ועד סופו בדומה לקובץ סדרתי רגיל. המערכת לניהול קבצים פונה רק לגושים של הנתונים ולא פונה אל האינדקסים, מאחר וכל גושי הנתונים משורשרים ביניהם. המשתמש מודיע למערכת במשפט SELECT על כך שברצונו לפנות לקובץ באופן סדרתי ולא לפי אינדקסים.

```
SELECT MLAY-FILE ASSIGN TO XX
    ORGANIZATION IS INDEXED
    RECORD KEY IS MISP-CATALOGI
    ACCESS MODE IS SEQUENTIAL.
```

הצהרת שיטת הפניה הסדרתית

בתוך התוכנית הקריאה היא רציפה כמו בקובץ סדרתי כלשהו והמשתמש מקבל את הרשומות בסדר לוגי עוקב. בסיום הקובץ עוברת התוכנית לקטע SOF.

```
PROCEDURE DIVISION.
        = = =
    OPEN INPUT MLAY-FILE.
        = = =
    READ MLAY-FILE AT END
            PERFORM SOF.
        = = =
    CLOSE MLAY-FILE.
        — — —
```

פניה אקראית מאפשרת גישה מהירה לרשומה כלשהי בתוך הקובץ. על המשתמש להגדיר את המפתח הדרוש ומערכת ניהול הקבצים מבצעת את הסריקות באינדקסים לאיתור הרשומה. יש לציין במשפט SELECT שהפנייה לקובץ היא אקראית.

```
SELECT MLAY-FILE ASSIGN TO XX
     ORGANIZATION IS INDEXED
     RECORD KEY IS MISP-CATALOGI
     ACCESS MODE IS RANDOM.          הצהרת פנייה אקראית לקובץ
```

```
PROCEDURE DIVISION.
     OPEN I-O MLAY-FILE.
     = = =
     MOVE "12347" TO MISP-CATALOGI.
     READ MLAY-FILE INVALID KEY                קריאת הרשומה
          GO TO NOT-FOUND.
     PERFORM IDKUN-KAMUT.                      עדכון
     REWRITE MLAY-REC INVALID KEY              כתיבת הרשומה במקומה
          PERFORM ABNORMAL.                    לאחר העדכון
       = = =
     CLOSE MLAY-FILE
     — — —
```

בדוגמא זו התוכנית פונה בגישה ישירה לרשומה כלשהי, מבצעת בה עדכון של כמות המלאי וכותבת אותה חזרה אל הקובץ באותו מקום. אם הרשומה לא נמצאת בקובץ מקבלת התוכנית הודעה על כך מהמערכת לניהול קבצים ופונה לקטע NOT-FOUND. אם יש תקלה בעת כתיבת הרשומה המעודכנת עוברת התוכנית לקטע ABNORMAL.

במערכות לניהול קבצים של יבמ על מנת לגשת לרשומה בצורה אקראית יש להגדיר שדה בשטח העבודה ולהעביר אליו את המפתח של הרשומה המבוקשת. שדה זה משמש כארגומנט החיפוש.

```
SELECT MLAY-FILE ASSIGN TO IS-3310-D-MLAY
        ORGANIZATION IS INDEXED
        RECORD KEY IS MISP-CATALOGI
        NOMINAL KEY IS NOM-KEY.
= = =
```

הצהרת שימוש בשדה מפתח נוסף לגישה אקראית

```
FD  MLAY-REC
    LABEL RECORD IS STANDARD
    DATA RECORD IS MLAY-REC.

01  MLAY-REC.
    03 DELETE-CODE          PIC X.
    03 MISP-CATALOGI        PIC 9(9).
    03 — — —
= = =

WORKING-STORAGE SECTION.
77 NOM-KEY PIC 9(9).
= = =
```

הגדרת שדה ארגומנט החיפוש

```
PROCEDURE DIVISION.

    OPEN I-O MLAY-FILE.
    = = =

    MOVE "123476" TO NOM-KEY.
    READ MLAY-FILE INVALID KEY
         GO TO NOT-FOUND.
    = = =

    CLOSE MLAY-FILE.
```

העברת המפתח לשדה
קריאה לפי מפתח

פניה דינמית מאפשרת שימוש בפניה אקראית וסדרתית באותה תוכנית. צורת פניה זו שימושית מאד כאשר יש לגשת לרשומה מסוימת בקובץ ולהתחיל לקרוא אותו באופן סדרתי מנקודה זו. בצורה זו ניתן לחסוך את הסריקה הסדרתית של הקובץ עד לרשומת ההתחלה. בהוראת SELECT יש להכריז על שיטת העיבוד של הקובץ.

בדוגמא זו מתבצעת גישה לרשומה עם מפתח נתון וקריאה סדרתית של כל הקובץ מנקודה זו.

```
SELECT MLAY-FILE ASSIGN TO XX
        ORGANIZATION IS INDEXED
        RECORD KEY IS MISP-CATALOGI
        ACCESS MODE IS DYNAMIC.        הצהרה של שיטת הפנייה
                                       לקובץ
PROCEDURE DIVISION.

    = = =
    OPEN INPUT MLAY-FILE.
    = = =
    MOVE "98765" TO MISP-CATALOGI.
    READ MLAY-FILE INVALID KEY         קריאה של רשומה
            GO TO NOT-FOUND.           לפי מפתח

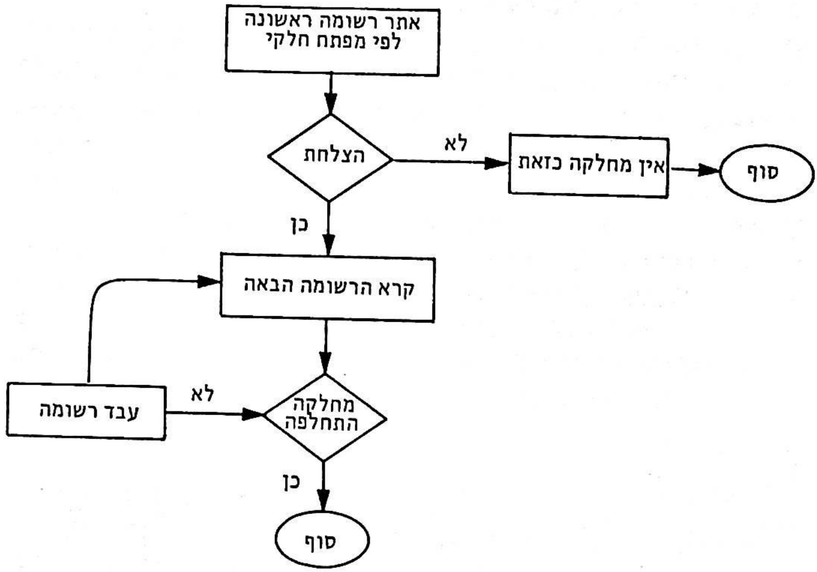
    = = =
    READ MLAY-FILE AT END              קריאה רצופה
            PERFORM SOF.               של רשומות

    = = =
    CLOSE MLAY-FILE.
```

שימוש במפתח חלקי

האינדקסים הממוינים לפי מפתח מאפשרים איתור רשומות בקובץ לפי מפתח חלקי בלבד (Generic Key או Major Key). המפתח החלקי חייב להיות החלק ה"מוביל" של המפתח. בדוגמה של מפתח המורכב מהאותיות ABCD יכול המפתח להיות A, AB או ABC. הצירוף BC לא יכול להיות מפתח חיפוש לאיתור רשומה.

השימוש במפתח חלקי מאפשר גמישות רבה בתכנון הגישות לקובץ. נניח שיש לנו קובץ פרויקטים לפי מחלקות. המפתח בנוי ממספר מחלקה ומספר פרויקט. כדי לקבל את רשימת כל הפרויקטים המנוהלים במחלקה מסוימת נוכל לאתר בקובץ את מספר המחלקה בלבד ואחר כך להתחיל לקרוא באופן סדרתי את כל הרשומות. פעולת האיתור לפי מפתח חלקי אינה מעבירה את הרשומה לשטח הרשומה ולכן יש לבצע מיד אחר כך פעולת קריאה.



תרשים 13-5: תרשים זרימה לשימוש במפתח חלקי

```
SELECT PROJECT-FILE ASSIGN TO YY
     ORGANIZATION IS INDEXED
     RECORD KEY IS FILE-KEY                          שדה המפתח
     ACCESS MODE IS DYNAMIC.                         הצהרת שיטת הגישה
= = =
FD  PROJECT-FILE
     LABEL RECORD IS OMITTED
     DATA RECORD IS PROJECT-REC.
01   PROJECT-REC.
     03   FILE-KEY.                       מפתח הרשומה מורכב משני חלקים:
          05   DEPARTMENT        PIC 99.                 מחלקה
          05   PROJ-NUM          PIC 9(5).               מספר הפרויקט
     03   PROJECT-DATA           PIC x (30).
     = = =
PROCEDURE DIVISION.
     = = =
     OPEN INPUT PROJECT-FILE.
     = = =
```

MOVE 82 TO DEPARTMENT.	
START PROJECT-FILE KEY NOT LESS FILE-KEY INVALID KEY GO TONOT-FOUND.	איתור רשומה ראשונה
READ PROJECT-FILE INVALID KEY PERORM ABNORMAL.	קריאת רשומה ראשונה
.	
LOOP.	
PERFORM TIPUL.	
READ PROJECT-FILE AT END PERFORM SOF.	קריאה סדרתית של הקובץ עד סופו
IF DEPARTMENT = 82 GO TO LOOP.	בדיקת תנאי
= = =	

ניהול מיקום הרשומה בקובץ

בקובץ הנמצא בדיסק ניתן לבצע גישה סדרתית וגישה ישירה באחת מהשיטות שהוזכרו. על מנת לאפשר ביצוע פקודות קריאה סדרתיות לאחר ביצוע פקודות קריאה אקראיות פועלת המערכת באמצעות מציין מקום (CRP — Current Record Pointer) המכיל את כתובת הרשומה הבאה אחרי הרשומה שכבר נקראה. רשומה זו תוצג למשתמש אשר מתחיל בקריאה סדרתית. למציין־מקום אין כל משמעות בעיבוד אקראי. תוכנו נקבע ע"י פקודות OPEN, START, READ בלבד, ושאר הפקודות אינן משנות את תוכנו.

לדוגמא נניח שהקובץ מכיל 30 רשומות עם מפתחות מ־1 ועד 30.

הפקודה בתוכנית	תוכן מציין־מקום	הערות
OPEN INPUT ...	רשומה ראשונה בקובץ	
=		
"העבר 10 למפתח"		
=		
START ...	10	
READ	11	הרשומה העשירית מועברת לתוכנית
READ NEXT	12	הרשומה האחד־עשרה מועברת לתוכנית
WRITE	12	
"העבר 28 למפתח"		
START	28	
READ	29	רשומה 28 מועברת לתוכנית
READ NEXT	סוף קובץ	רשומה 29 מועברת לתוכנית
READ NEXT		התוכנית ב־AT-END

אחזקת קובץ אינדקס סדרתי

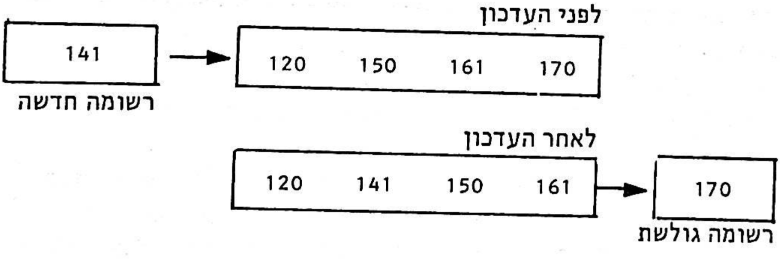
עד עתה ראינו כיצד בנית האינדקס מאפשרת גישה מהירה לרשומות בקובץ מבלי לאבד את היתרונות שבאפשרות עבודה סדרתית. עתה נסביר כיצד מבוצעת אחזקת הקובץ, כלומר הטיפול בהוספות, שינויים וביטולים של רשומות.

בעיות אחזקת הקובץ מתעוררות בגלל הצורך לשמור על התכונה הסדרתית של הקובץ גם לאחר ביצוע עדכונים. הוספת רשומה חדשה למשל, צריכה להתבצע בגוש מסוים במקומה הסידורי כדי שהקובץ ישאר ממוין. הבעיה היא שלא תמיד יש מקום בגוש להוספת הרשומה החדשה. בהמשך נציג שתי שיטות עיקריות לאחזקת קובץ אינדקס סדרתי:

— שטחי גלישה,
— פיצול דינמי של גושים.

שטח גלישה (Overflow Area)

בשיטה זו בונים שטחי גלישה מיוחדים לרשומות שאין להן מקום בגוש ומשתמשים במצביעים כדי לשמור על הסדר הלוגי של הקובץ. כדי להוסיף רשומה חדשה מאתרים את המקום בגוש שבו הרשומה צריכה להיות מאוחסנת בהתאם למפתח שלה. מאחר והגוש מלא ברשומות, יש לפנות מקום לכתיבת הרשומה החדשה. פינוי המקום מתבצע על־ידי הזזת כל הרשומות עם מפתח גדול מהמפתח של הרשומה החדשה במקום אחד. ההזזה גורמת ל"גלישה" של הרשומה האחרונה בגוש. רשומה זו מועברת לשטח אחסון אחר אשר נקרא **שטח גלישה**.



תרשים 5-14: דוגמא לגלישת רשומה מגוש הנתונים

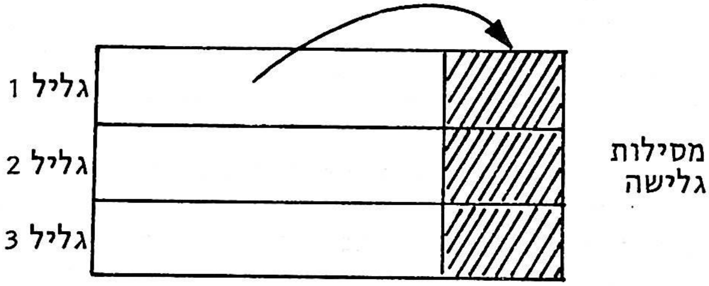
המשתמש אינו מרגיש בגלישה והמערכת לניהול קבצים אינה מודיעה לו שישנה גלישה. המערכת מטפלת בגלישות באופן עצמאי ונושא זה "שקוף" לחלוטין מבחינת המשתמש.

קיימות מספר שיטות לניהול שטחי גלישה:

— **גלישה במסגרת המסילה,**
— **גלישה במסגרת הגליל,**
— **שטח גלישה בלתי תלוי במסילה או בגליל,**
— **שיטה משולבת.**

גלישה במסגרת המסילה: בשיטה זו מגדירים **בכל מסילה** גוש נוסף אחד או יותר ומשאירים אותם **ריקים** בזמן טעינת הקובץ. רשומה שגולשת מאחד הגושים במסילה מועברת אל שטח הגלישה באותה המסילה. שיטה זו אינה גמישה מאחר והיא מאפשרת מספר מצומצם יחסית יש רשומות גולשות ולכן גידול הקובץ מוגבל למדי. אם ידוע מראש שמספר הרשומות החדשות הינו קטן יחסית ניתן להשתמש בשיטה זו. חסרון נוסף הוא בכך שהגלישה אינה מתפלגת באופן אחיד על פני הקובץ ולכן יתכנו מסילות שבהן ינוצל כל שטח הגלישה, או אף יחסר שטח גלישה, בעוד שבמסילות אחרות לא תהיה גלישה כלל.

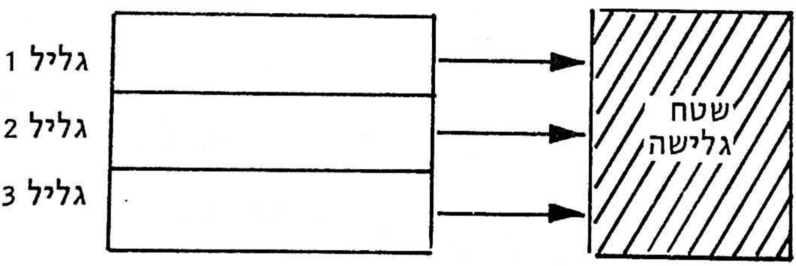
גלישה במסגרת הגליל: **בכל גליל** מגדירים מסילה אחת או יותר למטרת גלישה ומשאירים אותן ריקות בזמן טעינת הקובץ. רשומה גולשת מועברת אל **מסילת הגלישה** (Overflow Track).



תרשים 5-15: מסילות גלישה

כדי לפנות לרשומה הנמצאת במסלול גלישה אין צורך לבצע הזזת זרוע. הגדרת מספר גדול מדי של מסלולי גלישה יכולה לגרום לבזבוז רב של מקום איחסון. כמו־כן, הבעיה היא שפיזור הרשומות הגולשות אינו אחיד, כלומר קיימים גלילים בהם יש מספר מועט של רשומות גולשות בעוד שגלילים אחרים אין מקום לרשומות גולשות מפאת פיזור בלתי אחיד שלהן על פני הקובץ.

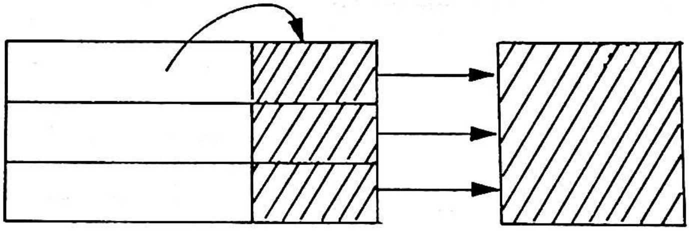
שטח גלישה בלתי תלוי (Independent Overflow Area): בשיטה זו מגדירים שטח גלישה אחד משותף לכל הקובץ. אין חשיבות מאיזה גוש או מסילה מגיעה הרשומה הגולשת. הבעיה בשיטה זו היא שגישה לרשומה



תרשים 5-16: שטח גלישה משותף

גולשת מחיבת הזזת זרוע ולכן הביצועים שלה גרועים יותר כאשר יש מספר רב של גלישות. לעומת זאת, שיטה זו גמישה יותר, מאחר ואינה תלויה בפיזור הרשומות הגולשות לפי הצילינדרים או הגושים. מאחר וכל הרשומות גולשות לשטח משותף יש לדאוג ששטח זה יהיה מספיק גדול לקליטת כל הרשומות הללו.

שיטה משולבת (Combined Overflow): שיטה זו מהווה שילוב בין שתי שיטות קודמות, הגלישה בגליל ושטח גלישה בלתי תלוי. מגדירים מסלול אחד או יותר לגלישה במסגרת הגליל כדי למנוע הזזות תכופות של הזרוע. כאשר אין יותר מקום בגליל מסוים, מועברות הרשומות הגולשות לשטח גלישה משותף.



תרשים 5-17: גלישה בשיטה משולבת

שרשור רשומות גולשות בקובץ אינדקס סדרתי

מבחינה לוגית הקובץ חייב להמשיך להיות ממוין לפי המפתח גם כאשר יש בו מצב של גלישה. כדי לקיים זאת יש צורך לשרשר את הרשומות הגולשות בצורה שתאפשר לסרוק את הקובץ בסדר עוקב. השרשור מתבצע על־ידי שימוש במצביעים הקושרים רשומה אל הרשומה הבאה מבחינה לוגית. דוגמא לשרשור רשומות בשטח גלישה משותף ניתן בתרשים 5-18.

I — אינדקס

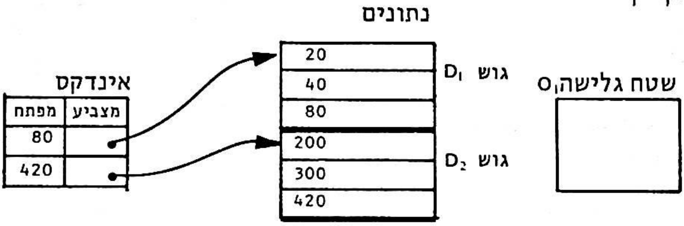
D — שטח הנתונים

O — שטח גלישה משותף

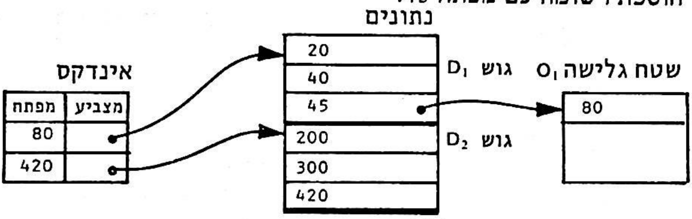
תרשים 5-18/1 מציג את מצב הקובץ לאחר טעינתו הראשונית. תרשים 5-18/2 מציג את מצב הקובץ לאחר הוספת רשומה עם מפתח 45. נשים לב לכך שבאינדקס לא חל שינוי מאחר והרשומה בעלת המפתח הגדול ביותר היא עדיין 80. מגוש D_1 נוצר מצביע לגוש הגלישה O_1 המציין קיום רשומה גולשת. תרשים 5-18/3 מציג את מצב הקובץ לאחר הוספת רשומה עם מפתח 10. המצביע לגלישה בגוש D_1 מורה עכשיו על הרשומה השניה בשטח הגלישה מאחר וכל הרשומות השיכות לגוש צריכות להיות ממוינות. הרשומה עם מפתח 45 הנמצאת בשטח הגלישה מצביעה על הרשומה עם מפתח 80, שהיא האחרונה בשרשרת של גוש D_1, תרשים 5-18/4 מציג את מצב הקובץ לאחר הוספת רשומה עם מפתח 480. לרשומה זו המפתח הגדול ביותר בגוש D_2 ולכן מתבצע

שינוי מתאים גם באינדקס. הרשומה עם מפתח 47 מתווספת בשטח הגלישה ומקושרת אל הרשומות 45 ו־80 שבשטח זה — תרשים 5-18/5.

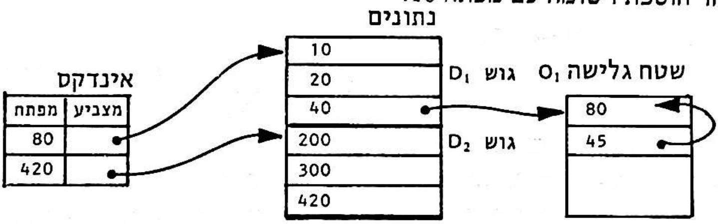
1. מצב הקובץ בתחילת העיבוד:



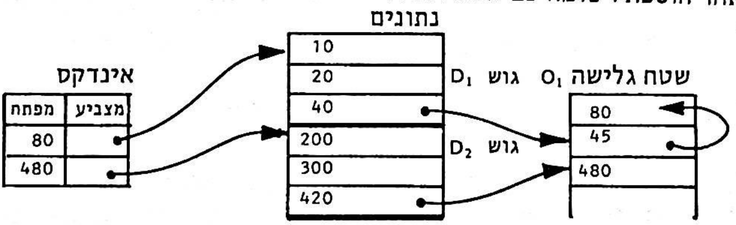
2. לאחר הוספת רשומה עם מפתח 45:



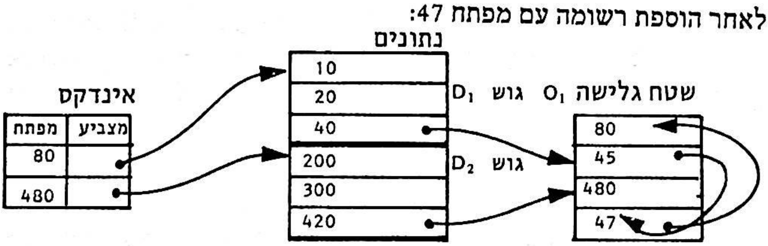
3. לאחר הוספת רשומה עם מפתח 10:



4. לאחר הוספת רשומה עם מפתח 480:



5. לאחר הוספת רשומה עם מפתח 47:



תרשים 5-18: שרשור רשומות בשטח גלישה משותף

שיטה זו לניהול גלישות מיושמת במערכות יבמ בשיטת ISAM.

בשיטת ISAM האינדקס מכיל שני סוגי מצביעים בכל רשומה:

מצביע רגיל (Normal) — מכיל את המפתח והכתובת לרשומה הגבוהה ביותר הנמצאת באותה מסילה.

מצביע גלישה (Overflow) — מכיל את המפתח והכתובת לרשומה הגבוהה ביותר השייכת לאותה מסילה אך נמצאת בשטח גלישה.

בנוסף קיים מצביע למקום הפנוי הבא בשטח הגלישה.

בזמן טעינת הקובץ שני המצביעים זהים, מאחר ואין גלישה לאף גוש. נשתמש בדוגמא המפורטת בתרשים 5-19.

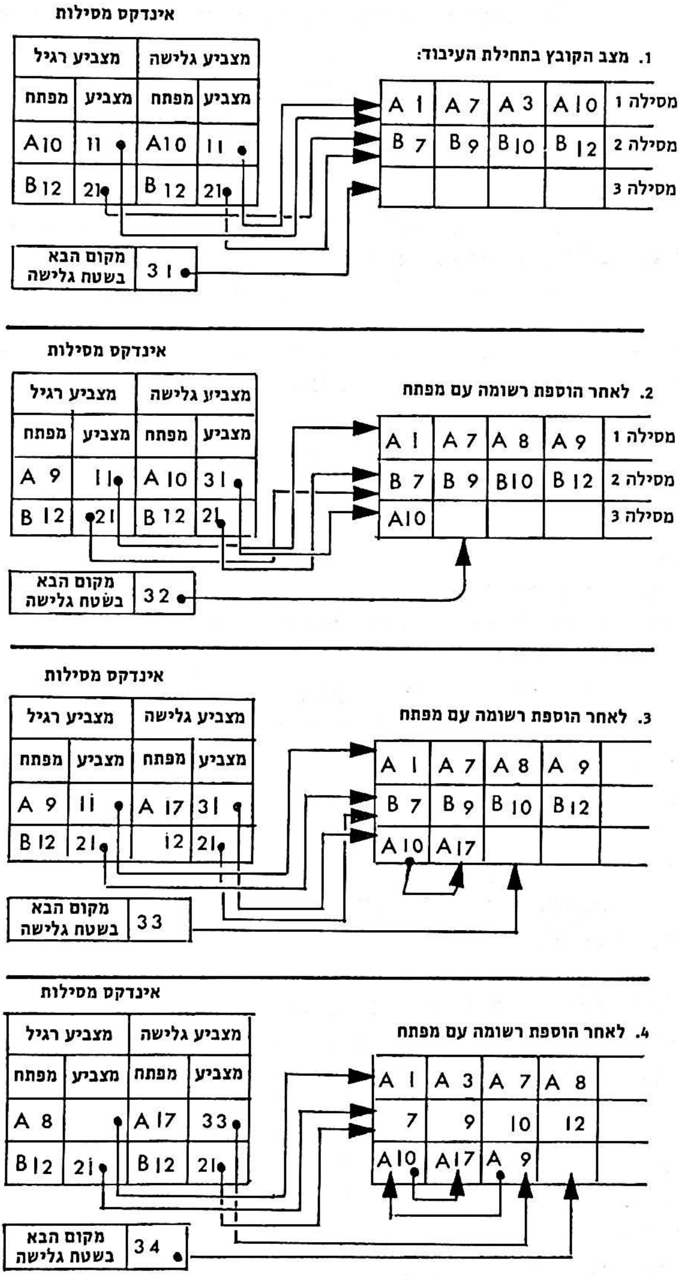
תרשים 5-19/1 מציג את המצב ההתחלתי של הקובץ לאחר טעינתו הראשונית. קובץ הנתונים תופס שתי מסילות. מסילה שלישית מיועדת לקליטת רשומות גולשות. כל כתובת בשטח נתונים מורכבת ממספר המסילה והמספר הסדורי בתוך המסילה. לדוגמא הכתובת של רשומה B10 יהיה 23 (מסילה 2, מקום 3). במצב ההתחלתי, האינדקס מצביע על התחלת המסילה ומכיל את המפתח הגבוה ביותר במסילה.

תרשים 5-19/2 מציג את מצב הקובץ לאחר הוספת רשומה עם מפתח A9. הרשומה החדשה גורמת לגלישת רשומה A10. הרשומה הגולשת תועבר לכתובת 31 (בהתאם לתוכן מצביע המקום הפנוי הבא בשטח הגלישה). האינדקס הרגיל יעודכן מאחר והמפתח הגבוה ביותר במסילה הוא A9. באינדקס הגלישה יש לעדכן את תוכן המצביע כך שיצביע על הכתובת של רשומה A10.

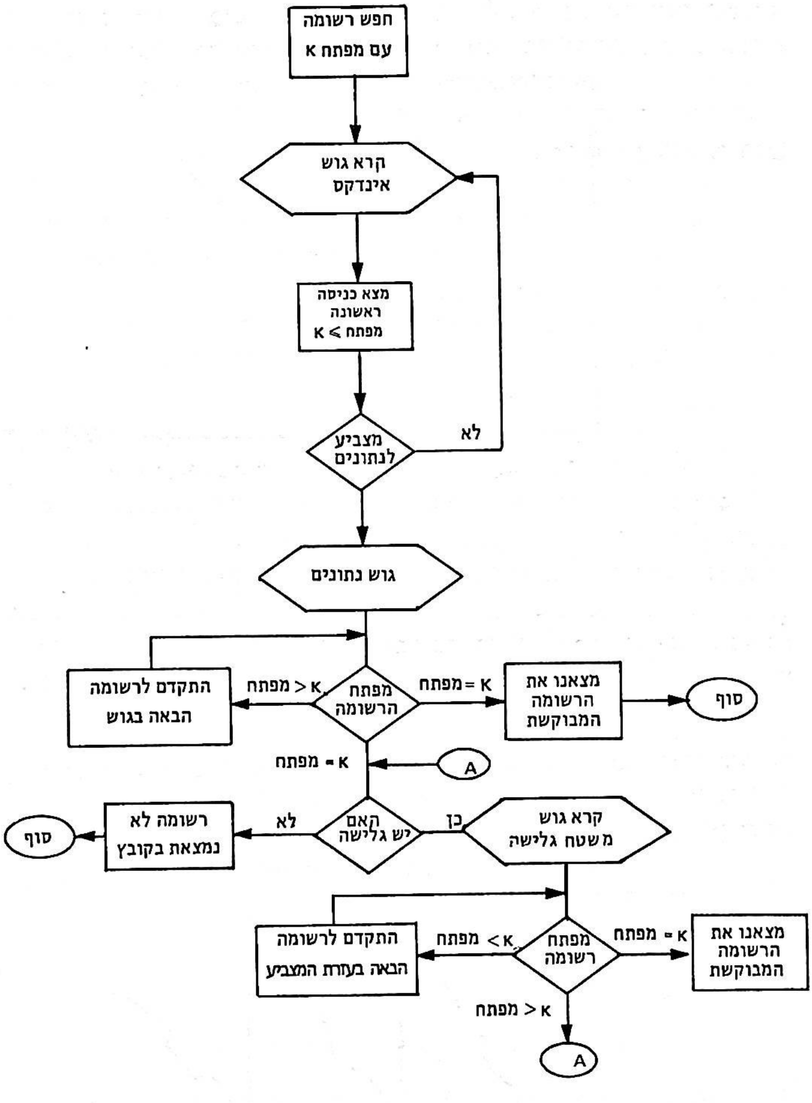
תרשים 5-19/3 מציג את מצב הקובץ לאחר הוספת רשומה עם מפתח A17, הרשומה החדשה מתווספת ישירות לשטח הגלישה לכתובת 32 (בהתאם לתוכן מצביע המקום הפנוי הבא בשטח הגלישה). הרשומות בשטח הגלישה משורשרות ביניהן כך שישמר המיון של הרשומות. אינדקס הגלישה מעודכן מאחר והמפתח הגבוה ביותר בגלישה הוא A17. המצביע ממשיך להצביע לרשומה A10 מאחר והיא הראשונה מבחינה לוגית בשטח הגלישה.

תרשים 5-19/4 מציג את מצב הקובץ לאחר הוספת רשומה עם מפתח A3. הרשומה החדשה גורמת לגלישת רשומה A9. רשומה זו מועברת לכתובת המתאימה בשטח הגלישה ומשורשרת עם שאר הרשומות הגולשות לשמירת סדר המיון. האינדקס הרגיל מתעדכן מאחר והמפתח הגבוה ביותר במסילה הוא עתה A8. אינדקס הגלישה מתעדכן מאחר והרשומה הראשונה, מבחינה לוגית, הוא A9.

עד כאן תיארנו את הטיפול בהוספת רשומות חדשות. כאשר מבטלים רשומות קיימות בקובץ רוב המערכות המשתמשות בשטחי גלישה רק מסמנות את הרשומות המבוטלות אבל אינן משחררות את המקום שהן תופסות. המקום שלהן משתחרר רק כאשר רשומה מבוטלת גולשת מהגוש או בזמן ביצוע ארגון מחדש של הקובץ.

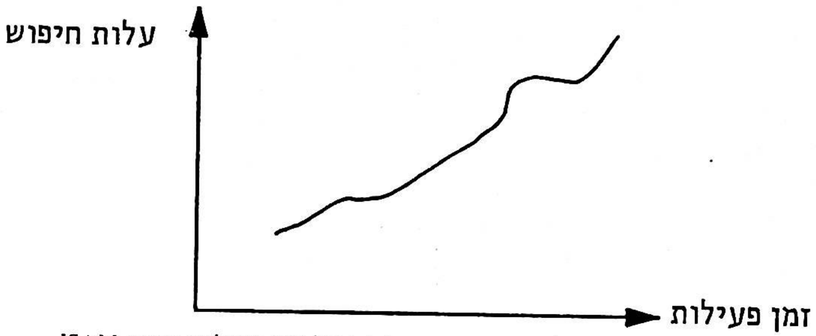


תרשים 5-19: עדכון קובץ ISAM של יבמ



תרשים 5-20: תרשים זרימה לחיפוש רשומה בקובץ ISAM עם גלישות

לאחר זמן פעילות מסוים, חלק גדול מהקובץ יכול להמצא בשטחי הגלישה לא לפי סדר המפתחות אלא בצורה משורשרת. במקרים אלה הגישה לרשומה מאד לא יעילה בגלל הצורך בהזזת זרועות רב מאד לקריאת מספר רב של רשומות בודדות. הביטולים גורמים גם הם לניצול לא טוב של שטחים, ולמעשה לבזבוז. אם נסמן ב־C את עלות חיפוש הרשומה, ניווכח שעלות החיפוש הולכת וגדלה ככל שמשך פעילות הקובץ ארוך יותר.

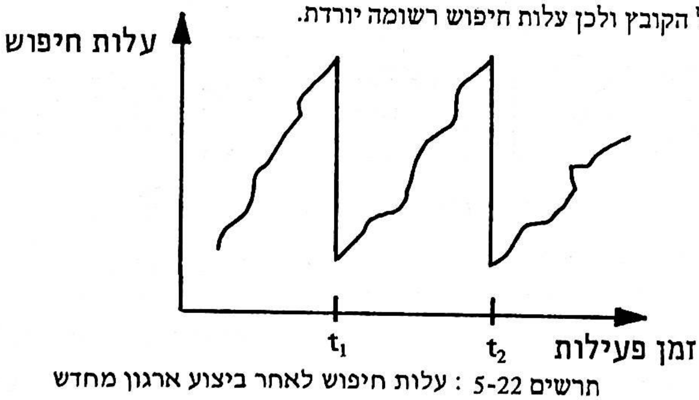


תרשים 5-21: עלות חיפוש כפונקציה של זמן פעילות קובץ ISAM

המשתמש יכול לגלות את **העליה בעלות החיפוש** מתוך סטטיסטיקות המנוהלות על־ידי המערכת לניהול קבצים, כגון — שיעור התפוסה בשטח הגלישה, מספר הפניות לרשומות בשטחי גלישה, ומספר הרשומות המבוטלות בקובץ.

על המשתמש **לארגן מחדש את הקובץ על־ידי טעינה מחדש** (Load) בעזרת תוכנית שירות. לאחר הארגון מחדש של הקובץ מתקבל מצב התחלתי ופונקציית עלות החיפוש תהיה שונה מבמקרה הקודם.

תרשים 5-22 מציג את הגידול בעלות חיפוש רשומה כפונקציה של זמן הפעילות. ככל שמשך הפעילות גדול יותר, מספר הרשומות בשטחי גלישה יהיה גדול יותר ולכן עלות חיפוש רשומה תגדל. בנקודת זמן t_1 מתבצע ארגון מחדש של הקובץ ולכן עלות חיפוש רשומה יורדת.



תרשים 5-22 : עלות חיפוש לאחר ביצוע ארגון מחדש

פיצול דינמי של גושים

שיטת הפיצול הדינמי של גושים (Dynamic Block Splitting) פותחה כדי להתגבר על בעית הירידה בביצועים בשיטת ISAM (כלומר, משך זמן ארוך יותר לגישה לרשומה) כפונקציה של משך פעילות הקובץ. בשיטה זו אין שטחי גלישה והקובץ מורכב משני שטחים:

— **איזור נתונים** המורכב מגושים המכילים את הרשומות. להבדיל מהשיטה הקודמת, המשתמש יכול להגדיר בזמן טעינת הקובץ אחוז מסוים משטח הגוש כשטח ריק (Padding). שטח זה ישמש לקליטת רשומות חדשות בתוך הגוש המקורי ללא צורך בביצוע פעולות נוספות.

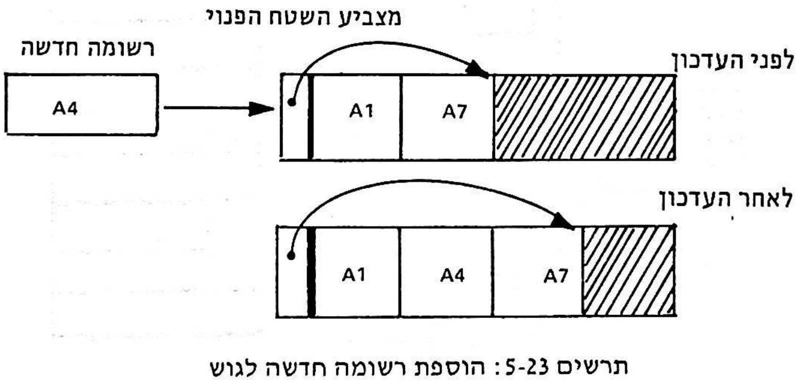
קיימות מערכות המקצות את כל איזור הנתונים בעת טעינת הקובץ כדי לקבל מספר רב ככל האפשר של גושים קרובים אחד לשני (Bulk Allocation). לעומתן, קיימות מערכות המקצות גושים בהתאם לצורך ומוסיפות גושים תוך כדי גידול הקובץ.

— **איזור אינדקסים** — מורכב מגושים של אינדקסים. גם בגושים אלה ניתן להגדיר אחוז מסוים משטח הגוש כשטח ריק בזמן טעינת הקובץ.

אחזקת הקובץ

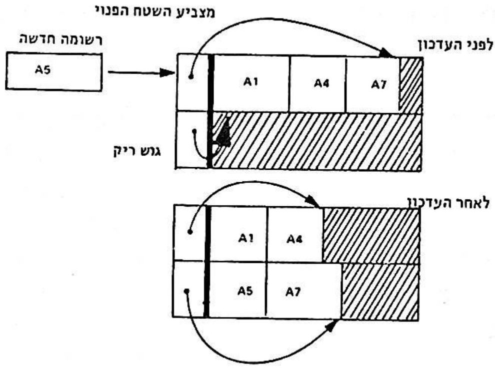
קיימים שני מצבים בזמן הוספת רשומה חדשה:

1. **יש מקום פנוי בגוש** — במקרה זה הרשומה החדשה תתווסף בגוש המקורי במקום הלוגי המתאים לה ויתבצע עדכון של מצביע בתוך הגוש המורה להתחלת השטח הפנוי. הוספת רשומה במצב זה יכולה לגרום לעדכון המפתח הגבוה ביותר בגוש אבל לא לעדכון הכתובת, מאחר והגוש החדש נכתב חזרה במקום הגוש הקודם (שים לב שאנו פועלים במתקן אחסנה המאפשר זאת — דיסק מגנטי).



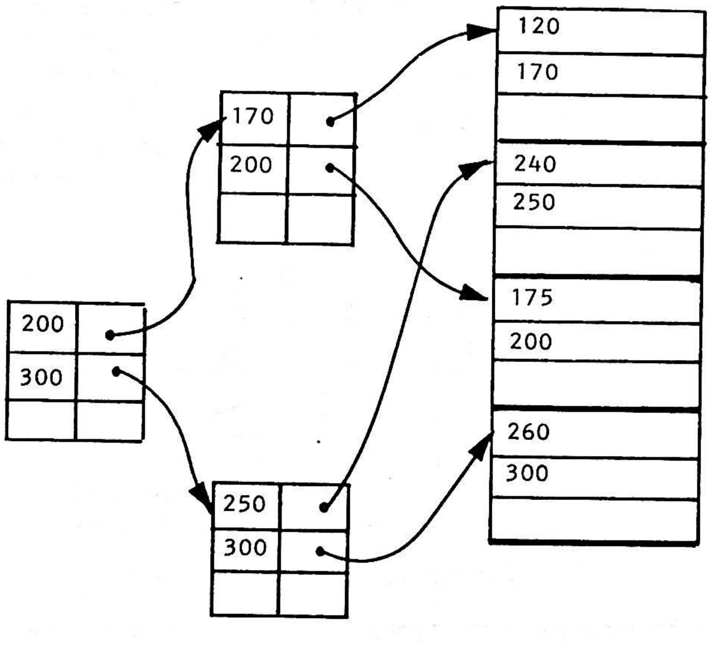
תרשים 5-23: הוספת רשומה חדשה לגוש

2. **אין מקום בגוש** לקליטת הרשומה החדשה, ואז מתבצע פיצול של הגוש. חצי ממספר הרשומות נשארות בגוש המקורי וחצי ממספר הרשומות מועתקות לגוש ריק חדש. הגוש החדש אינו חייב להיות שכן לגוש המקורי, מאחר והסדר הלוגי של הגושים נשמר באינדקסים. בתוך הגוש הרשומות ממוינות תמיד.



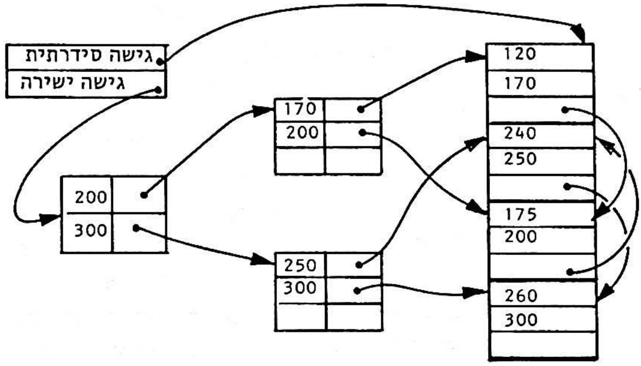
תרשים 5-24: פיצול גוש לשני גושים בעת עדכון

בתרשים 5-25 ניתנת דוגמא המתארת את הצורה שבה מתעדכנים האינדקסים בעקבות פיצול גושים.



תרשים 5-25: הוספת רשומה עם מפתח 260 ופיצול אינדקסים

על מנת לאפשר קריאה סדרתית של הקובץ ללא צורך בפניות לאינדקסים ניתן לשרשר ביניהם את כל הגושים לפי הסדר הלוגי שלהם כפי שמתואר בתרשים 5-26.



תרשים 5-26: שרשור סדרתי של גושים בקובץ אינדקס סדרתי

5.4 ארגון אינדקס סדרתי
(VSAM — Virtual Storage Access Method)

VSAM היא שיטה מתקדמת לארגון נתונים במערכות יבמ המשתמשות במערכות הפעלה עם זיכרון בפועל (Virtual Storage). ב-VSAM מבחינים בשני סוגי קבצים בהתאם לסדר הרשומות:

— **קובץ עם רשומות ממוינות** (Key Sequenced Data Set — KSDS). בסוג זה הרשומות ממוינות בהתאם למפתח כלשהו, ו-VSAM מנהל עבורו אינדקס בדומה לשיטת ארגון אינדקס סדרתי.

— **קובץ עם רשומות לא ממוינות** (Entry Sequenced Data Set — ESDS). בסוג זה הרשומות נרשמות בקובץ בהתאם לסדר הגעתן ללא כל התיחסות לתוכן הרשומות או למפתח כלשהו. עבור סוג זה אין VSAM מנהל אינדקס כלשהו. על שיטת הגישה לרשומות אלו נעמוד בהמשך.

שיטת אחסון רשומות ב-VSAM

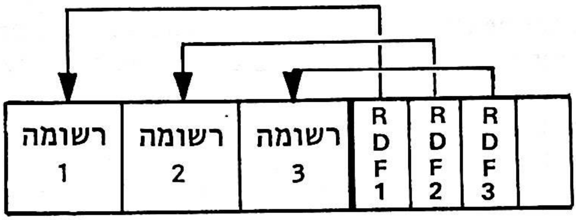
VSAM מאחסן את כל הרשומות בדיסק בשטח בעל אורך קבוע. זהו שטח הקרוי **גוש הפיקוח** (Control Interval). הוא נבדל מהגוש הפיסי אותו הכרנו עד כאן בכך שזהו **מבנה לוגי**. לכן גוש פיקוח אחד יכול להיות כלול במספר גושים פיסיים, אם כי בדרך כלל יש להשתדל לקבוע את גודלו בגודל זהה לגוש הפיסי. גודל גוש הפיקוח נמצא בשליטת המשתמש וניתן לקביעה לפני בנית הקובץ. קיימת גם אפשרות ש-VSAM יחשב את גודל הגוש בעת טעינת הקובץ.

רשומה 1	רשומה 2	רשומה 3	נתוני בקרה

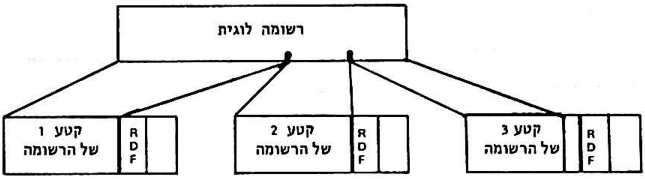
תרשים 5-27: מבנה גוש הפיקוח

הרשומות הפיסיות נרשמות בתחילת גוש הפיקוח ואילו בסופו נרשמים נתוני בקרה הנמצאים בשימוש VSAM בלבד. נתונים אלו מכילים שדה אחד (RDF — Record Definition Field) עבור כל רשומה בגוש הפיקוח אשר מורכב מאורך הרשומה ונקודת ההתחלה שלה בגוש.

קיימת אפשרות לפרוס רשומה פיסית אחת על פני יותר מגוש פיקוח אחד אם הרשומה הפיסית ארוכה מאד.



תרשים 5-28: שדות בקרה בגוש הפיקוח



תרשים 5-29: פריסת רשומה לוגית במספר גושי פיקוח

אוסף של גושי פיקוח קרוי **תחום פיקוח** (Control Area). מספר הגושים המרכיבים את תחום הפיקוח הינו קבוע ונקבע על־ידי המשתמש או על־ידי VSAM. בדרך כלל השאיפה היא שתחום הפיקוח יהיה גליל אחד.

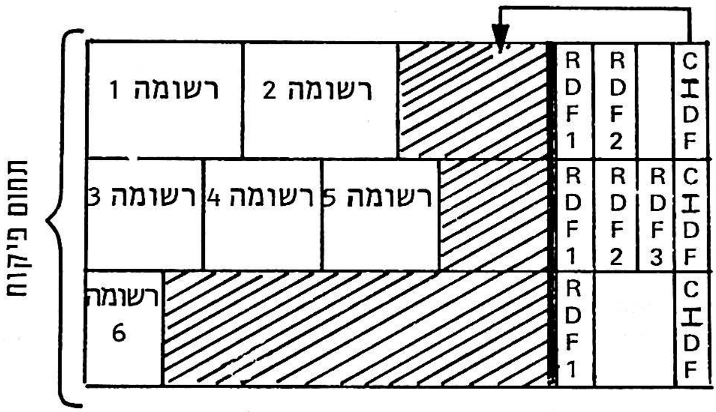


תרשים 5-30: מבנה תחום פיקוח ב־VSAM

פיזור שטח חופשי

VSAM משתמש בשיטת הפיצול הדינמי של הגושים לטיפול בעדכונים. על מנת למנוע את הצורך בפיצול הגוש מיד עם הוספת הרשומה הראשונה לגוש, משתמש VSAM בשיטה של פיזור שטח ריק (Free-Space Distribution) בתוך הגוש ובתוך תחום הפיקוח. המשתמש יכול להגדיר מראש מהו אחוז שטח הגוש או תחום הפיקוח שיש להשאיר ריק בזמן בניית הקובץ. התוצאה היא שהשטח המוקדש לגלישות בשיטת ISAM, פוזר כאן בתוך הגושים השונים כבר בשלב בניית הקובץ.

כדי לנהל את השטח הריק בתוך הגוש משתמש VSAM בשדה בקרה הקרוי CIDF (Control Interval Definition Field) המתאר את גודל השטח הריק בגוש ואת נקודת התחלתו.

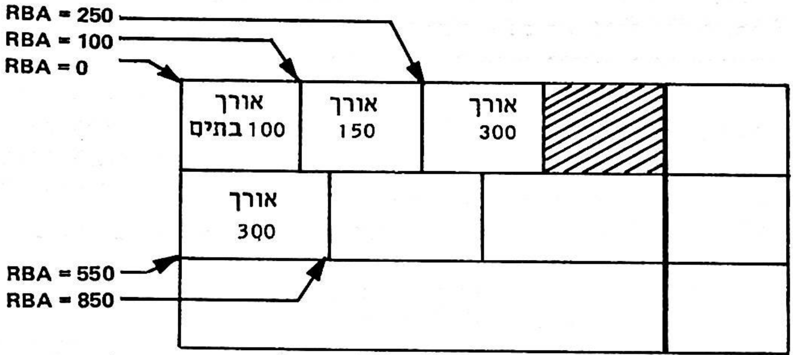


תרשים 5-31: פיזור שטח חופשי בגוש

חלק מהגושים המרכיבים את תחום הפיקוח נשארים ריקים בזמן בניית הקובץ. כאשר מתבצע פיצול של גוש, נלקח גוש ריק מתוך תחום הפיקוח. לאחר שנוצלו כל הגושים בתחום הפיקוח, מבצע VSAM הקצאה אוטומטית של תחום פיקוח חדש.

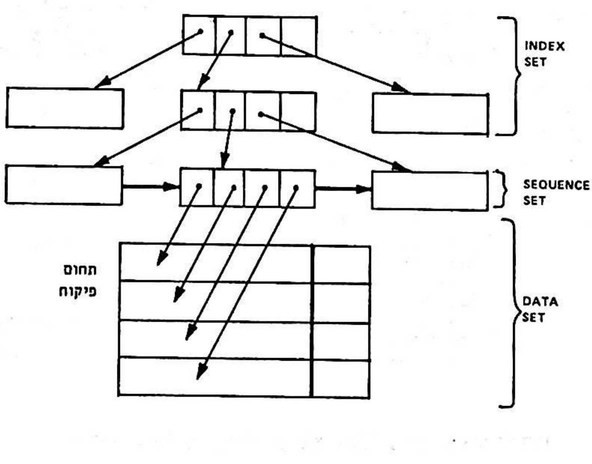
שיטות הגישה לרשומה -VSAM

הפניה לרשומות ב־VSAM אינה מתבצעת על פי תכונות פיזיות כגון מספר הגליל ומספר המסילה (כמו ב־ISAM) אלא **על פי היסט** (Displacement) הרשומה, בבתים, מתחילת הקובץ. הכתובת היחסית של הרשומה קרויה RBA (Relative Byte Address). הכתובת היחסית של הרשומות אינה משתנה בעקבות העברת הקובץ לדיסק אחר או למקום אחר באותו הדיסק.



תרשים 5-32: הכתובת היחסית של רשומות ב־VSAM

כאשר מגדירים קובץ KSDS בונה VSAM באופן אוטומטי אינדקס המאפשר גישה מהירה לרשומות. האינדקס עצמו מורכב מרשומות אינדקס המסודרות במספר רמות. הרמה התחתונה נקראת Sequence Set והיא מכילה רשומה אחת עבור כל תחום פיקוח בקובץ. הרשומה מכילה מספר שדות (כמספר הגושים בתחום הפיקוח) ובהם ערך המפתח הגבוה ביותר בכל אחד מהגושים בתחום הפיקוח. VSAM מתיחס אל האינדקס כאל קובץ נפרד מקובץ הנתונים. תיאור סכימטי של מערך האינדקסים ותחום הפיקוח ניתן בתרשים 5-33.

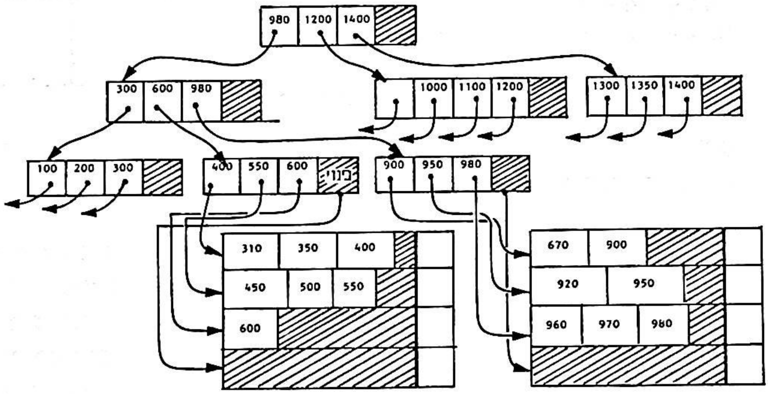


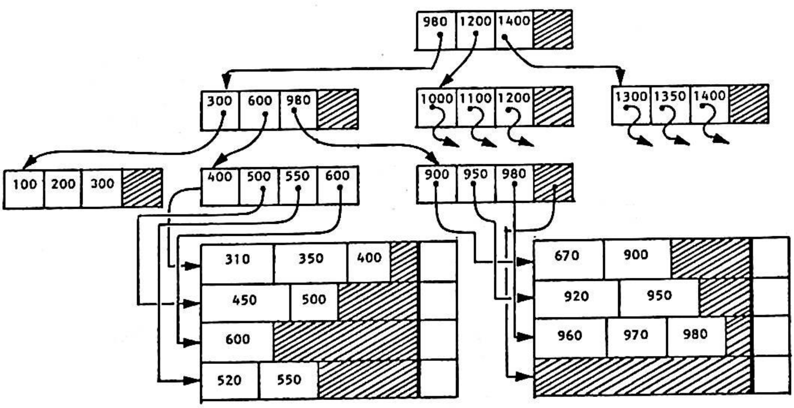
תרשים 5-33: מבנה קובץ KSDS

רשומות האינדקס ברמה הנמוכה ביותר משורשרות ביניהן כדי לאפשר סריקה סדרתית של הקובץ ללא פניות לכל רמות האינדקס. בדרך כלל רשומת האינדקס הזאת נמצאת במסילה הראשונה של תחום הפיקוח והיא מוחזקת מספר פעמים להקטנת זמן ההשהיה הסיבובית.

האינדקסים ברמות הגבוהות יותר מצביעים על רשומות באינדקס בהתאם למפתח הגבוה ביותר. תרשים 5-34/1 מציג את תוכן האינדקסים בקובץ KSDS.

1. מבנה קובץ VSAM במצב התחלתי:

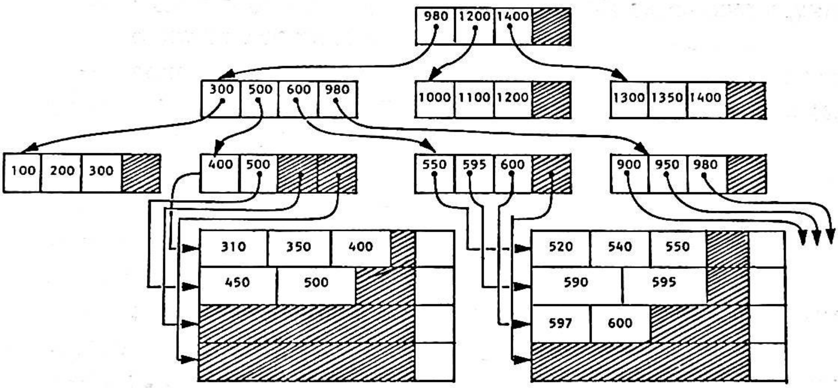




תרשים 5-34: עדכון קובץ VSAM-KSDS של יבמ

כאשר מוסיפים רשומות חדשות לקובץ, VSAM מנסה להכניס אותן לשטח הריק בתוך הגוש בהתאם למפתח. תרשים 5-34/2 מציג דוגמא להוספת רשומה במצב זה.

כאשר אין מקום בתוך הגוש להוספת הרשומה החדשה, מתבצע פיצול של הגוש. חלק מהרשומות מועברות לגוש חדש וחלקן נשאר בגוש המקורי. VSAM מעדכן באופן אוטומטי את האינדקסים בהתאם. לאחר הפיצול שוב יש שטח פנוי בשני הגושים כך שהוספת רשומה חדשה לאחד מהגושים לא תביא בהכרח לפיצול נוסף.



תרשים 5-35: פיצול גוש מקורי לשני גושים

כאשר מגדירים קובץ מסוג ESDS אין VSAM בונה כל אינדקס. לאחר כתיבת רשומה חדשה, VSAM מחזיר למשתמש את הכתובת היחסית של הרשומה (RBA) כדי שינהל בעצמו את הכתובות בשיטת אינדקס כלשהי אשר הוא קובע. האינדקסים מאפשרים את הגישה לרשומה מאוחר יותר. שיטה זו דומה לשיטת הארגון הסדרתי אם אין צורך לגשת לרשומות לפי מפתח. VSAM גם מאפשר למשתמש לכתוב את הרשומות לכתובות הנקבעות על ידו מראש ובכך מאפשר גישה ישירה לרשומות.

5.5 סיכום

יתרונות:

— אפשרות לעיבוד סדרתי ואקראי כאחד.
— ניצול טוב של שטח האיחסון מאחר ובדרך כלל לא מתבצעת הקצאה מראש של שטח האחסון.
— המערכת לניהול קבצים מספקת מספר שירותים, כגון: תוכניות שירות לבנית קובץ אינדקס סדרתי, סטטיסטיקות על מספר רמות האינדקס, ניצול שטחי גלישה.

חסרונות:

— ב־ISAM יש שטחי גלישה מיוחדים המהווים בזבוז שטחי אחסון מאחר ולא תמיד הם מנוצלים במלואם. במשך הזמן יש ירידה בביצועים בגלל פיזור רשומות בשטחי גלישה וצורך בארגון מחדש.
— ב־VSAM זמן הוספת רשומה גדול יותר בגלל הצורך בפיצול גוש נתונים ו/או פיצול גוש אינדקסים.
— בשתי השיטות הגישה לרשומה מחייבת סריקת אינדקסים אשר משפיעה על זמן התגובה ועל משך העיבוד.
— האינדקסים תופסים שטחי אחסון נוספים לשטחי הנתונים.

5.6 שאלות

1. הסבר את ההבדלים בין ISAM ל-VSAM.
2. קובץ מכיל רשומות באורך 500 בתים. גודל הגוש 3000 בתים, גודל המפתח 30 בתים, גודל שדה הכתובת 4 בתים, בקובץ 500,000 רשומות. חשב את מספר רמות האינדקס הדרושות.
3. הסבר כיצד משתמשים בגושים להקטנת גודל האינדקס.
4. הסבר את מושג המפתח החלקי. כיצד ניתן להשתמש בו. הבא מספר דוגמאות לשימוש במפתח חלקי.
5. מדוע יש צורך בארגון מחדש של קובץ ISAM? מדוע דרוש ארגון מחדש בקובץ VSAM?
6. קובץ מכיל גושים באורך 3000 בתים עם רשומות באורך 1000 בתים. הראה את המבנה הסכימטי של קובץ אינדקס סדרתי המכיל את הרשומות עם המפתחות הבאים.

10, 17, 20, 80, 90, 104, 112, 117, 120, 121, 128

150, 160, 170, 175, 180, 191, 198, 201, 301, 404, 600

הצג את מבנה האינדקס ושטחי הגלישה לאחר הוספת הרשומות הבאות:

8 ,604, 21 ,185, 155

7. קובץ ISAM נמצא בדיסק בעל הנתונים הבאים:
   — 200 מסילות, 10 משטחי מגנוט (ראשי קריאה כתיבה),
   — זמן חיפוש ממוצע — 30 מילישניות,
   — השהייה סיבובית — 8 מילישניות,
   — מהירות העברה — 0.8 מיליון בתים לשניה.
   גודל הקובץ — 50,000 רשומות, 50 רשומות למסילה.
   חשב:
   — מספר הגלילים הדרוש לאחזקת הקובץ,
   — זמן הגישה הממוצע לרשומה.

5.7 נספח — חישוב מספר רמות אינדקס

נתון קובץ שבו גודל גוש הוא B בתים (או מילים) וגודל הרשומה הוא R. לפיכך יהיו בכל גוש B/R רשומות. אם בקובץ יש n רשומות, נקבל שמספר רשומות האינדקס הדרושות יהיה:

$$\text{מספר רשומות אינדקס} = \frac{n}{B/R} = n\frac{R}{B}$$

רשומת אינדקס מכילה את המפתח שגודלו K וכתובת לגוש המתאים, שגודלה P ולכן האורך של רשומת אינדקס יהיה K + P.

נניח שגודל הגוש של האינדקס גם הוא B (תנאי לא הכרחי, יתכן גודל גוש שונה) לכן מספר רשומות האינדקס שניתן להכניס לגוש האינדקס יהיה

$$\text{מספר רשומות בגוש אינדקס} = \frac{B}{P+K}$$

נסמן ב־L את מספר רמות האינדקס. בכל רמת אינדקס ניתן להצביע על $\frac{B}{P+k}$ גושים, שבכל אחד מהם ניתן שוב להצביע על $\frac{B}{P+K}$ גושים וכך הלאה.

$$\left(\frac{B}{P+K}\right)^L = n\frac{R}{B}$$

$$\boxed{L = \frac{\log\left(n\frac{R}{B}\right)}{\log\left(\frac{B}{P+K}\right)}}$$

מאחר ומספר הרמות חייב להיות מספר שלם יש לקחת את המספר השלם הגדול הקרוב ביותר ל־L.

פרק שישי

ארגון נתונים לגישה ישירה
(Direct Access Method)

בפרק זה נציג את שיטות הארגון המאפשרות גישה ישירה לרשומה ללא צורך בסריקת אינדקסים.

6.1 מבוא

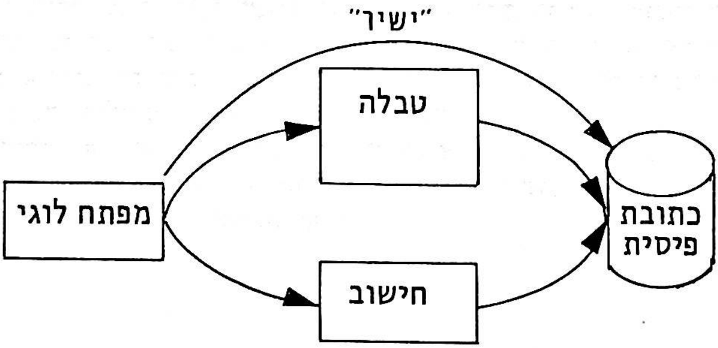
בפרק הקודם הצגנו את השיטה המשתמשת באינדקסים לאיחזור רשומות. הקיום של יחידות האיחסון לגישה ישירה הובילה באופן טבעי לנסיונות לארגן את הנתונים בצורה שתמנע את הצורך בסריקת אינדקסים ותאפשר גישה ישירה לרשומה המבוקשת.

הארגון לגישה ישירה (Direct Access Method) הינו שיטה לארגון נתונים בה קיים קשר, ידוע וקבוע מראש, בין המפתח של הרשומה לבין הכתובת הפיסית שלה בקובץ. שיטה זו מאפשרת גישה מהירה מאד לרשומות לצרכי שאילתות ועדכונים בהתאם למפתח המבוקש. לתהליך הפיכת המפתח לכתובת פיסית נקרא **תרגום** (Key-To-Address Transformation).

קיימות מספר שיטות לביצוע התרגום של המפתח לכתובת פיסית:

— יחוס ישיר (Direct Addressing)
— יחוס לפי טבלה (Table Lookup)
— יחוס על־ידי חישוב (Calculated Addressing)





תרשים 6-1: קשר ישיר בין מפתח למיקום הרשומה בקובץ

6.2 יחוס ישיר

בשיטה זו מתאימים את הכתובת הפיסית בצורה חד־ערכית למפתח, כלומר הכתובת שווה למפתח. רשומה בעלת מפתח K נמצאת בקובץ בכתובת K. זוהי כמובן כתובת יחסית להתחלת הקובץ ועל המערכת לניהול קבצים לתרגם כתובת יחסית זו לכתובת מוחלטת בדיסק. המפתחות יכולים להיות רק מספרים שלימים חיוביים. אם המפתח הוא אלפא־ביתי יש לבצע תרגום לערך מספרי כלשהו.

לדוגמא, אם מספר העובד הוא 191 הרשומה שלו תימצא במקום היחסי 191 בקובץ. בשיטה זו ניהול מיקום הרשומות בקובץ הינו באחריות המשתמש. ראה בנספח כיצד ניתן להפוך כתובת יחסית לכתובת מוחלטת.



תרשים 6-2: יחוס ישיר של מפתח לכתובת פיסית

היתרון של השיטה הזאת הוא היחס הפשוט יחסית בין המפתח לכתובת ולכן היא מאפשרת גישה מהירה מאד לרשומה המבוקשת. בשיטה זו אין צורך לנהל שטחי גלישה מאחר ולכל רשומה שמור מראש מקום בקובץ. העובדה שיש לשמור לכל רשומה מקום מראש היא גם החסרון הגדול של השיטה. אם טווח הערכים שהמפתח יכול לקבל הוא גדול מאד ומספר הרשומות בקובץ הוא קטן, נקבל נצילות גרועה מאד של שטח האיחסון.

לדוגמא, אם נשתמש במספר קטלוגי בן 9 תווים כמפתח הרי שטווח הערכים שלו הוא בין 0 ל־10^9-1 כלומר 10^9 ערכים שונים. אם יש בקובץ 200,000 רשומות בלבד, הנצילות תהיה:

$$\frac{2 \cdot 10^5}{10^9} = 0.02\%$$

נצילות זו נמוכה מאד, ולכן יש להשתמש במקרה זה בשיטת ייחוס שונה. לסיכום שיטת היחוס הישיר נציין את היתרונות והחסרונות שלה.

יתרונות:

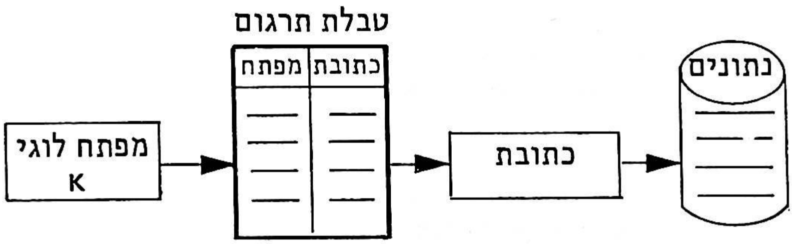
— שיטה יעילה מאד אם טווח המפתחות הוא קטן יחסית וצפיפות הרשומות גדולה.
— גישה מיידית לרשומה לפי קשר פשוט בין המפתח לכתובת הפיסית.
— ניתן לעבד את הקובץ גם באופן סדרתי מאחר והוא ממוין לפי המפתחות (יש להבחין בין מקומות תפוסים למקומות ריקים).
— עדכון הקובץ פשוט ביותר. לכל רשומה בקובץ יש מקום מוקצה מראש.

חסרונות:

— נצילות נמוכה מאד של שטח האחסון כאשר טווח המפתחות גדול יחסית למספר הרשומות האפשריות בקובץ. בדרך כלל זהו המקרה השכיח.
— השיטה מתאימה לרשומות באורך קבוע בלבד.

6.3 יחוס לפי טבלה

בשיטת הייחוס לפי טבלה נרשמות הרשומות בקובץ לפי סדר הקליטה. כלומר, רשומה חדשה תרשם בסוף הקובץ. לאחר כתיבת הרשומה בקובץ, מחזירה המערכת לניהול קבצים את הכתובת של הרשומה למשתמש. ניהול הכתובות לשם גישה לרשומות הינו באחריות המשתמש ולשם כך הוא בונה טבלה המבצעת את התרגום בין המפתח הלוגי לבין הכתובת הפיסית. בכדי לגשת לרשומה, יש לסרוק תחילה את הטבלה למציאת הכתובת.



תרשים 6-3: תרגום מפתח לכתובת בעזרת טבלה

שיטת הגישה באמצעות טבלה מהירה מאד מאחר והכתובת מאפשרת גישה מיידית לרשומה. מהירות הגישה לרשומה נקבעת על פי זמן חיפוש בטבלה המכילה מספר כניסות כמספר הרשומות בקובץ. אם מספר הרשומות רב, יש צורך לנהל אינדקסים לטבלה כדי ליעל את החיפוש. בעיית הגישה לקובץ הועברה לגישה לטבלה.

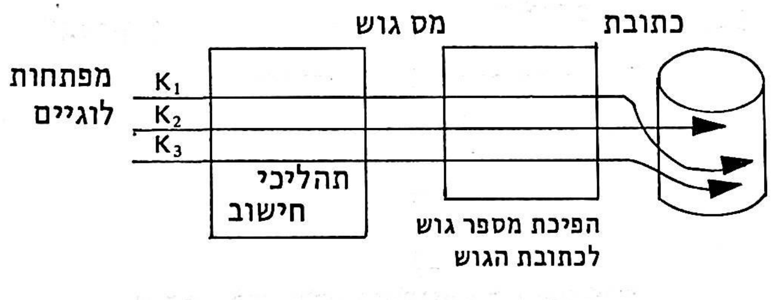
שיטה זו שימושית מאד כאשר יש לגשת לקובץ עם מספר רב של מפתחות בשיטות של אינדקס משני (ראה פירוט בפרק הבא). במקרה זה תנוהל הטבלה באופן אוטומטי על־ידי המערכת לניהול קבצים האחראית לניהול האינדקס המשני.

בדרך כלל רוב המערכות אינן משחררות שטח שהתפנה בעקבות ביטול רשומות. שטח זה מתפנה רק עם ארגון מחדש של הקובץ.

6.4 יחוס על־ידי חישוב

השיטה הנפוצה ביותר לתרגום המפתח לכתובת היא היחוס על ידי חישוב. השיטה מבוססת על הפעלת תהליך חישובי כלשהו המבצע את צימצום הטווח של הערכים שהמפתח יכול לקבל. בדוגמא שהבאנו ביחוס ישיר ראינו שטווח הערכים האפשריים הוא 10^9 בעוד שמספר הרשומות בקובץ הוא 200,000 בלבד. אם נבצע צימצום הטווח, לא נצטרך להקצות מראש מקום ל־10^9 רשומות ולכן נקבל ניצול טוב יותר של שטח האיחסון.

לתהליך החישובי המבצע את צמצום הטווח קוראים Hashing או Randomizing, והוא מתבצע בעזרת אלגוריתם כלשהו. מציאת פונקציה מתמטית שתבצע התאמה חד־חד־ערכית בין שני הטווחים היא בלתי אפשרית, ומכאן נובע החסרון של השיטה. התהליך החישובי יכול להתאים לכל מפתח כתובת פיסית או להתאים למספר מפתחות כתובת אחת, למשל את כתובת הגוש. הבעיה היא שהתהליך החישובי יכול להתאים למספר מפתחות את אותה הכתובת (Synonims) וכאשר מספר המפתחות האלה גדול ממספר הרשומות בגוש תתקבל גלישה. השיטה מטפלת בארגון הנתונים הגולשים מגושים ומסילות. כתוצאה מהגלישה נקבל ירידה בביצועים, כלומר זמן גישה ארוך יותר, מכיון שיש לבצע גישות נוספות לשם שליפת רשומות גולשות.



תרשים 6-4: תרגום מפתח לכתובת בתהליך חישובי

על האלגוריתם לפזר בצורה אחידה, עד כמה שניתן, את הרשומות בשטח האיחסון. ככל שפיזור הרשומות בקובץ יהיה טוב יותר נקבל מספר גלישות קטן יותר. אך נושא זה אינו בשליטת המשתמש והוא מותנה באופי הנתונים בלבד. מספר הרשומות הגולשות בלבד אינו מדד מספיק לבדיקת טיב הפיזור. מדד נוסף הינו מספר הגישות הכולל לשליפת כל הרשומות בקובץ.

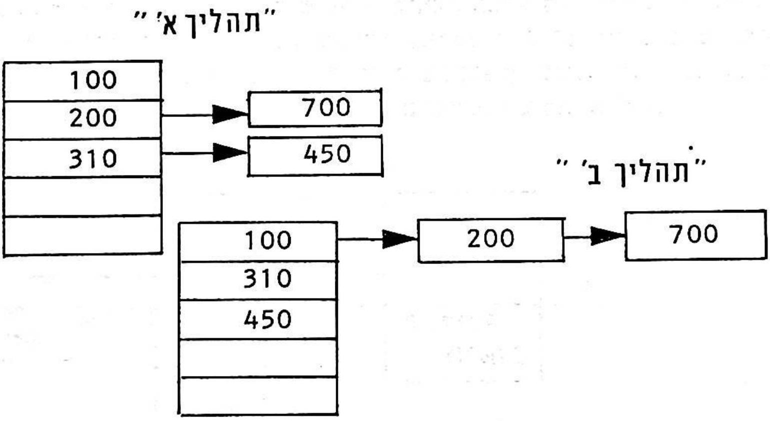


תרשים 6-5: דוגמא לקבלת כתובות זהות למפתחות שונות

נשתמש בדוגמא כדי להמחיש את הנושא. על סידרה של 5 מפתחות שונים מפעילים שני תהליכי תרגום שונים לקבלת הכתובות:

מפתח	תהליך א׳	תהליך ב׳
100	1	1
200	2	1
310	3	2
450	3	3
700	2	1

Synonims



תרשים 6-6: פיזור גלישות בשתי שיטות חישוב שונות

בשתי השיטות קיבלנו 2 רשומות גולשות. בתהליך א׳ ל־3 רשומות ניתן לגשת בגישה אחת ולשתי רשומות נוספות יש צורך בשתי גישות. כלומר, מספר גישות בקובץ יהיה $7 = 2 \times 2 + 3 \times 1$. בתהליך ב׳ ל־3 רשומות ניתן לגשת בגישה אחת. לרשומה אחת יש צורך בשתי גישות ולאחת יש צורך בשלש גישות. לכן מספר הגישות הכולל יהיה $8 = 3 \times 1 + 2 \times 1 + 3 \times 1$. המסקנה היא ששיטה א׳ עדיפה.

קיימים מספר גורמים המשפיעים על פיזור הרשומות בקובץ וטיב התרגום של התהליך החישובי.

א. **קיבולת הגוש.** ככל שגדלה קיבולת הגוש, מספר הרשומות הנכנסות לגוש אחד גדל ולכן הסתברות הגלישה קטנה יותר.

ב. **שיעור הדחיסה של הקובץ** (Packing Density) הוא היחס בין מספר הרשומות המשוער בקובץ למספר הרשומות האפשריות בקובץ. אם מספר הרשומות המשוער בקובץ הוא 60,000 והוקצה מקום ל־100,000 רשומות נאמר ששיעור הדחיסות של הקובץ הוא 0.6. ככל שהקובץ דחוס יותר ההסתברות לגלישה הולכת וגדלה. מצד שני, אם הקובץ "דליל" מאד ניצול שטח האיחסון גרוע יותר.

ג. **ערכים של המפתחות של רשומות הקובץ.** אם הערכים של המפתחות מתקבצים סביב ערכים מסוימים יביא התהליך החישובי לכך שנקבל מספר גלישות רב. לפני ההחלטה על הפעלת ארגון נתונים ישיר יש לבדוק את טיב הפיזור המתקבל בקובץ בעזרת התהליך החישובי. באופן תיאורטי אין אפשרות להגדיר תהליך חישובי שיוצר כתובות אקראיות ממפתחות שאינם מפולגים בצורה אקראית, בכל אופן קימת אפשרות למצוא תהליך חישובי שייצר כתובות די אקראיות.

מאחר ושיטת הייחוס על־פי חישוב הינה הנפוצה ביותר נרחיב בהמשך את הדיון בשיטה זו.

6.5 שיטות לחישוב כתובות

תהליך חישוב הכתובת מורכב בדרך כלל משני שלבים:

א. תרגום המפתח לערך מספרי שניתן להפעיל עליו תהליך חישובי כלשהו. אם המפתח מכיל מספר שלם, התרגום הינו פשוט ביותר. אם הוא מכיל גם אותיות או סימנים מיוחדים יש לקבוע שיטה לתרגום. ניתן להשתמש בצורה ישירה בייצוג הבינרי של האותיות בתור מספרים. אם בית אחד מכיל 8 סיביות ומייצג אות אחת ניתן לחלק אותו לשני חלקים של 4 ולהתייחס לערך מספרי המיוצג על ידי הסיביות.

ב. תהליך חישובי כלשהו אשר מטרתו היא צמצום הטווח של מספר הערכים האפשריים של המפתח לטווח מספר הרשומות הצפוי בקובץ. יש לבנות את התהליך החישובי כך שמספר המפתחות המייצרים את אותה כתובת יהיה מינימלי. נסמן ב־ h את פונקצית התרגום. פונקציה זו יכולה לקבל M ערכים שונים לכל היותר, כאשר M מספר הרשומות המקסימלי בקובץ.

$$o \leqslant h\,(K) \leqslant M$$

נציג בהמשך מספר שיטות לתרגום מפתח לכתובת.

חלוקה (Division)

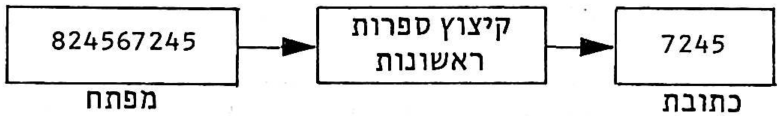
שיטת החלוקה הינה שיטה פשוטה ביותר. מחלקים את המפתח במספר הרשומות הצפויות בקובץ ומשתמשים בשארית ככתובת. לדוגמא, אם קובץ מכיל 1000 רשומות נחלק את המפתח הנתון 84256 ב־1000 ונקבל שארית 256 שהיא הכתובת של הרשומה בקובץ. זוהי כתובת יחסית שיש להפוך לכתובת מוחלטת לשם גישה לרשומה. מאחר ומחלקים ב־1000, השארית תנוע בין 0 ל־999 **וזהו בדיוק טווח מספר הרשומות האפשריות בקובץ.** אם הקובץ מכיל מפתחות רציפים הגדלים באחד, לאחר 1000 רשומות יתחיל אלגוריתם זה לייצר את אותן הכתובות.

$$h\,(K) = K \bmod M$$

ניתן להשיג שיפור ניכר אם המחלק יהיה מספר ראשוני הקרוב ביותר למספר הרשומות האפשריות בקובץ. שימוש במספר ראשוני מקטין את מספר הכתובות הזהות המיוצרות על־ידי התהליך החישובי. שיטה זו נחשבת לטובה מאחר והיא מייצרת גלישות מאוזנות.

קיצוץ (Truncation)

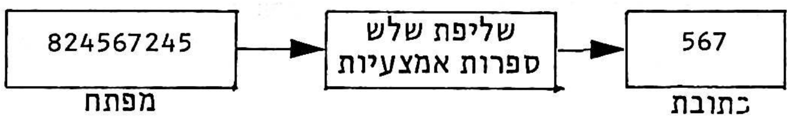
בשיטה זו מקצצים את הספרות הראשונות או האחרונות במפתח ויוצרים מפתח בעל מספר ספרות קטן יותר. מספר הספרות הנשאר נקבע על־ידי הטווח הרצוי. שיטה זו טובה כאשר קיים פיזור טוב של ערכי המפתחות.



תהליך הקיצוץ של 3 ספרות מימין מקביל לחלוקה ב־1000 אך מהיר ממנה מאחר ופעולת ההזזה מהירה יותר מאשר פעולת החילוק. אם הספרות האחרונות או הראשונות במפתח נוטות להתקבץ סביב ערכים קבועים נקבל גלישות רבות.

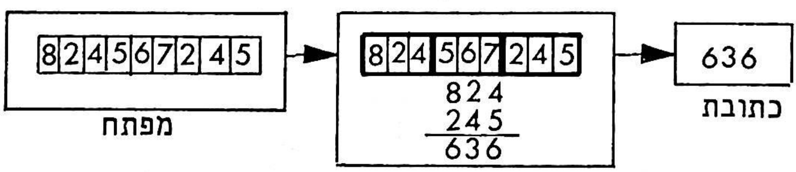
שליפה (Extraction)

שיטת השליפה דומה מאד לשיטת הקיצוץ אלא שכאן מבצעים קיצוץ משני צידי המפתח ושולפים את הספרות האמצעיות של המפתח בלבד. בחינת התפלגות המפתחות יכולה להוביל אותנו להחלטה אילו ספרות יש לשלוף. כמו בשיטה הקודמת, גם שיטה זו מאפשרת צמצום טווח בצורה פשוטה יחסית.



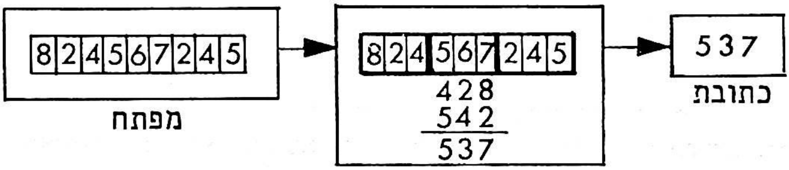
קיפול וחיבור (Folding/Adding)

בשתי השיטות הקודמות השתמשנו רקבחלק מהספרות המרכיבות את המפתח, אך כדי להשיג פיזור טוב יותר של הרשומות יש להשתמש בכל הספרות שלו. קיימות לכך מספר שיטות. בשיטה אחת "מקפלים" את המספר לשתיים או יותר חלקים ומחברים יחדיו את כולם. התוצאה של החיבור תשמש אותנו ככתובת הרשומה. לדוגמא, קיפול וחיבור כתובת כדי ליצור טווח של 1000 רשומות בתחום 0 עד 999.



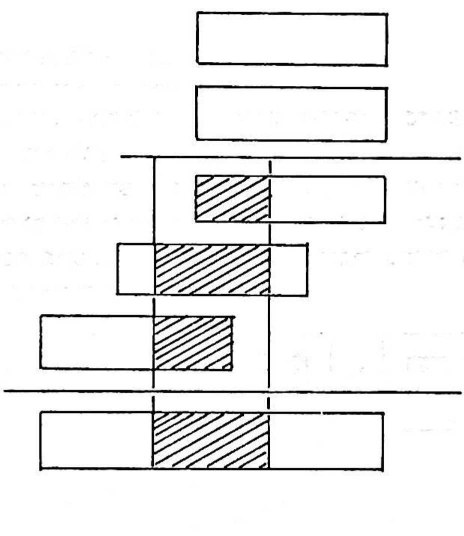
כאשר יש בקובץ 10,000 רשומות יש לייצר כתובות בטווח 0 ועד 9999. במקרה זה יש לחלק את המספר לשני חלקים בלבד.

קיימת גם אפשרות ל"סדר" מחדש את המספרים לפני החיבור להשגת אפקט נוסף של פיזור (Folding/Shifting). במקרה שבדוגמא נרשמו שלש הספרות הראשונות ושלש הספרות האחרונות בסדר הפוך.



ריבוע המפתח (Mid-Square Method)

בשיטה זו כופלים את המפתח בעצמו. ומתוך התוצאה המתקבלת שולפים את הספרות האמצעיות לקבלת הכתובת. אם חלקים גדולים מתוך המפתח יכולים להיות אפסים שיטה זו לא תייצר כתובות טובות. ה"מחיר" שבחישוב המפתח הוא גבוה אך יש לשקול זאת לעומת מספר הגישות הצפויות בקובץ וחסכון אפשרי בשטח אחסון על ידי פיזור טוב יותר של הרשומות בטווח האחסון.

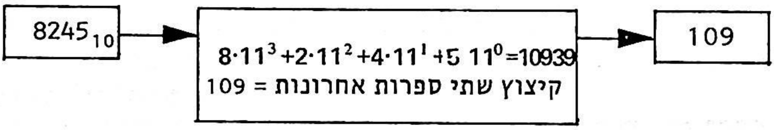


שינוי בסיס (Radix Transformation)

תהליך שינוי בסיס המספר של המפתח מתבצע בשלשה שלבים:

— הפיכת המפתח למספר בשיטת הייצוג העשרונית או הבינארית,
— שינוי הבסיס, כלומר הפיכת המספר מבסיס עשרוני למשל,לבסיס אחר.
— קיצוץ ספרות לקבלת הכתובת.

נדגים שיטה זו כאשר הבסיס החדש יהיה 11.



רצוי שהבסיס החדש יהיה מספר ראשוני. הבסיס החדש לא יכול להיות קטן מהבסיס המקורי (בבסיס 7 לא קיימת הספרה 8).

6.6 גישה ישירה ב־COBOL

מנקודת מבט המשתמש, השימוש בשיטת הגישה הישירה המוצגת **בקובול** דומה מאד לשימוש בארגון אינדקס סדרתי. נציין שניתן גם בשיטה זו לקרוא את הרשומות בצורה סדרתית, אבל הרשומות אינן ממוינות אלא מתקבלות בהתאם לסדר הפיסי שלהן.

המשתמש מגדיר את שיטת ארגון הקובץ, המפתח ומספר הגושים בקובץ במשפט SELECT. שאר ההגדרות והטיפול ברשומות הם בדומה לאלו של שיטת ארגון אינדקס־סדרתי.

```
SELECT MLAY-FILE ASSIGN TO XX
    ORGANIZATION IS DIRECT
    RECORD KEY IS MISP-CATALOGI
    NUMBER OF BLOCKS IS 101
    BLOCK CONTAINS 20 RECORDS
    ACCESS MODE IS RANDOM.
```

במערכות לניהול קבצים של יבמ השיטה שונה. על המשתמש לחשב בעצמו את מספר המסילה היחסית למסילה הראשונה. המסילות בקובץ ממוספרות מ־0 ועד n. המשתמש חייב לבנות מפתח (מחוץ לרשומה) המכיל את מספר המסילה ואת זיהוי הרשומה. המערכת מעבירה באופן אוטומטי מפתח זה אל הרשומה לפני שהיא מועברת ליחידת האיחסון. בעזרת TRACK-LIMIT קובע המשתמש מראש את מספר המסילות שהקובץ יתפוס.

בקריאה סדרתית של קובץ מאורגן בצורה ישירה, על המשתמש לזהות לבד אם הרשומה קיימת או שזהו מקום ריק.

נציג כאן דוגמא של תוכנית הבונה קובץ בגישה ישירה במערכות יבמ. בקובץ יש 8000 רשומות, וגודל המפתח 8 תווים. על מנת להשאיר מקום לרשומות גולשות נוסיף עוד 25% מקומות פנויים לכן נבנה מקום ל־10,000 רשומות. כל רשומה מכילה 200 בתים ונניח שהדיסק בנוי ממסילות היכולות להכיל 13 רשומות למסילה.

קביעת מסלול יחסי:

1. מספר רשומות למסילה הוא 13. לאיחסון 10,000 רשומות דרושות 770 מסילות.
2. המספר הראשוני הקרוב ביותר ל־770 הוא 691.
3. נחלק את המפתח ב־691. השארית מפעולת החילוק יהיה מספר המסילה היחסי.

תוכנית הקובול המתאימה:

```
SELECT MLAY-FILE ASSIGN TO DA-3330-D-MLAY
    ORGANIZATION IS DIRECT
    ACTUAL KEY IS ACT-KEY
    TRACK-LIMIT IS 691
    ACCESS MODE IS RANDOM.
————
————
WORKING-STORAGE SECTION.
77  QUOTIENT      PIC S9(9) COMP SYNC.
77  SAVE          PIC S9(9) COMP SYNC.
————
————
01  ACT-KEY.
    03  TRACK-ID    PIC S9(5)   COMP SYNC.
    03  RECORD-ID    PIC  9(9).

PROCEDURE DIVISION.
    OPEN OUTPUT MLAY - FILE.
    ——
    ——
    MOVE KEY TO SAVE RECORD-ID.
    DIVIDE SAVE BY 691 GIVING QUOTIENT
        REMAINDER TRACK-ID.
    WRITE MLAY-REC INVALID KEY
        GO TO ABN.
    ——
    ——
    CLOSE MLAY-FILE.
```

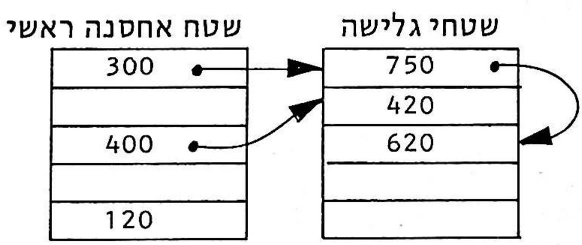
6.7 טיפול בגלישות

השימוש בתהליך חישובי לצמצום הטווח גרם ליצירת מספר כתובות זהות עבור מפתחות שונים ולכן יש לבנות מנגנון שיטפל ברשומות גולשות אלו. קיימות מספר שיטות לטיפול ברשומות גולשות — כאלה המקצות לרשומות הגולשות שטח נפרד בקובץ (בדומה לשיטת ISAM) וכאלה המאחסנות את הרשומות הגולשות יחד עם הרשומות הרגילות.

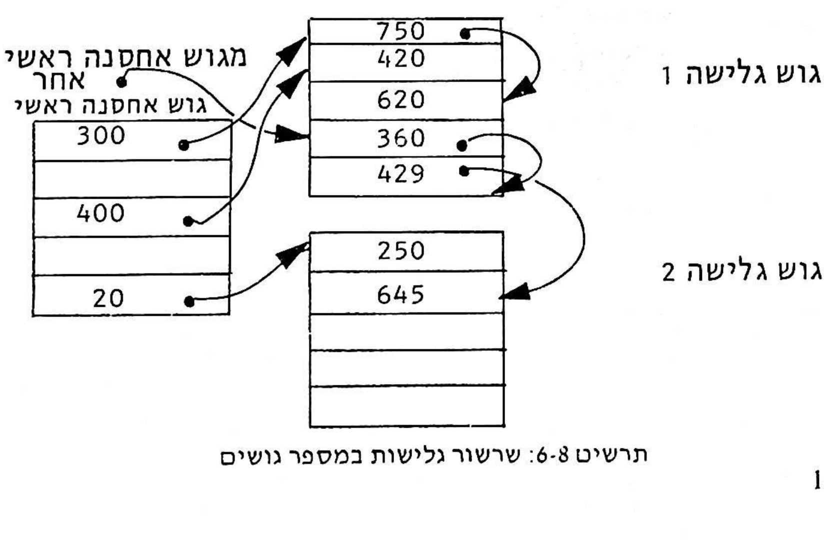
שטח גלישה נפרד (Overflow Chaining)

בזמן בניית הקובץ נגדיר שטח מיוחד שיקלוט את הרשומות הגולשות (Overflow Area), ונשרשר אותן על־ידי שימוש במצביעים.

דוגמא: נניח שהתהליך החישובי מפיק את אותה כתובת לרשומות עם מפתחות 300, 750, 620 וכתובת אחרת למפתחות 400, 420.



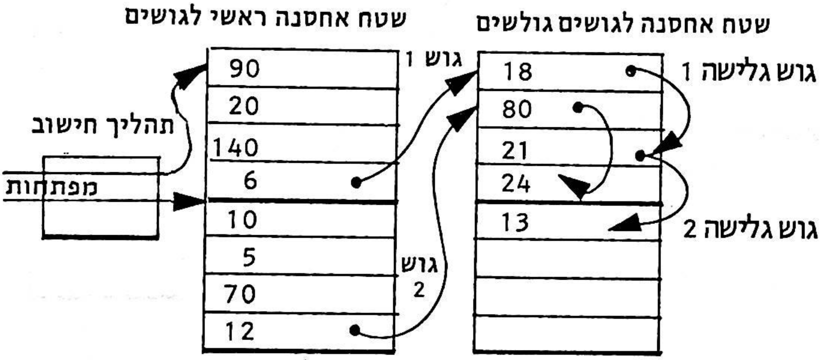
תרשים 6-7: שרשור רשומות גולשות



תרשיט 6-8: שרשור גלישות במספר גושים

הרשומה הראשונה נכנסת לכתובת המקורית ואילו כל הרשומות הגולשות מועברות לשטח הגלישה ומשורשרות ביניהן. ברגע שגוש הגלישה מתמלא, מתחילים למלא גוש חדש. יתכנו שרשרות הנמשכות על פני מספר גושים של גלישה.

בדרך כלל מבצעים את צמצום הטווח על־ידי התהליך החישובי לא למספר הרשומות האפשריות אלא למספר הגושים בקובץ (Home Blocks). כאשר מחשבים את הכתובת של הגוש, מביאים אותו לזכרון ומבצעים בו חיפוש סדרתי לאיתור הרשומה הדרושה. אם הרשומה לא נמצאת בגוש המקורי משתמשים במצביע אל שטח הגלישה.



תרשים 6-9: שרשור רשומות גולשות

בתרשים 6-9 רשומות 18, 21, 13 מיועדות לגוש 1. מאחר והגוש היה מלא הן הועברו לגוש גלישה 1. כאשר רשומה 13 הגיעה לגוש גלישה 1 גם הוא היה מלא ולכן היא הועברה לגוש גלישה 2. לסריקת כל הרשומות אשר הכתובת המחושבת שלהן היא 1 נצטרך במקרה זה שלש גישות מאחר והרשומות המיועדות לגוש זה מפוזרות עכשיו בשלשה גושים שונים.

נחשב את מספר הרשומות הגולשות מהגוש:

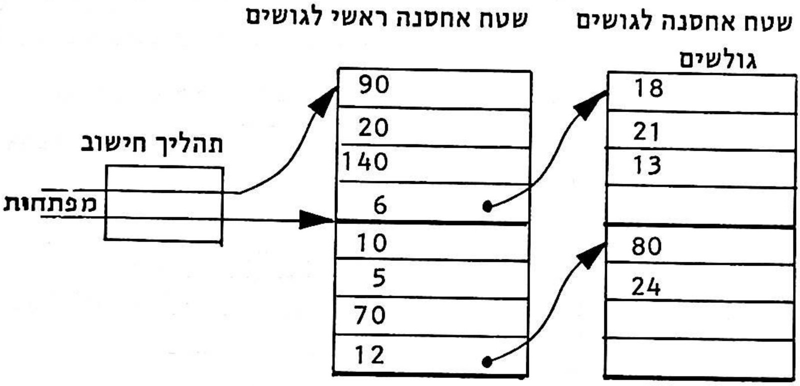
נסמן ב־N את מספר הרשומות בקובץ, ב־M את מספר הגושים בקובץ וב־b את מספר הרשומות בגוש. ההסתברות שמפתח כלשהו יכנס לגוש היא 1/M. ההסתברות שבדיוק X מפתחות יכנסו לאותו גוש תהיה:

$$P(X) = \frac{N!}{X!\ (N-X)!}\left(\frac{1}{M}\right)^{X}\left(1-\frac{1}{M}\right)^{N-X}$$

נחשב את מספר הרשומות הגדול מ־b המופנות לאותו גוש.

$$n = \sum_{x=b+1}^{N} P(X)\,(X-b)$$

בשיטה זו איתור הרשומה מתבצע תוך סריקת השרשרת. אפשרות אחרת היא לא לבנות שרשרת אלא רק מצביע לגוש הגלישה. לכל גוש בשטח הראשי שיש לו רשומות גולשות, יוקצה גוש גלישה נפרד, ויהיה לו מצביע אחד בלבד אל הגוש השייך לו. איתור הרשומה יבוצע על־ידי סריקת כל הרשומות בגוש הגולש עד לאיתור הרשומה המבוקשת. הסריקה נדרשת מכיון שלא ידוע איזו רשומה גלשה מאיזו כתובת בגוש הראשי.



תרשים 6-10: שטח גלישה נפרד ללא שרשור רשומות

שיטה זו קלה ליישום מאחר ואין שרשראות הקושרות את כל הרשומות שגלשו מאותה כתובת. ביטול הרשומה מהיר יותר מאחר ואין צורך לעדכן מצביעים. מצד שני זמן חיפוש הרשומה גדול יותר.

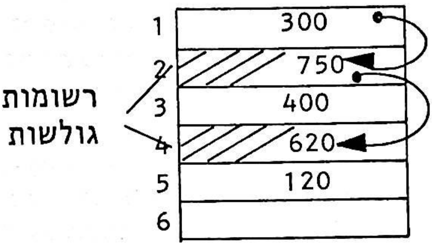
שטח גלישה בתוך השטח הראשי

בשיטה זו אין שטח מיוחד בקובץ לניהול רשומות גולשות אלא מנצלים מקומות פנויים בשטח הראשי. כדי לישם את השיטה חייב שיעור הדחיסות של הקובץ להיות קטן מ־100%.

קיימות שתי שיטות לניהול רשומות גולשות בשטח הראשי:

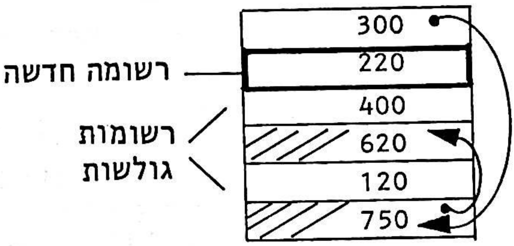
1. **שרשור** (Chaining) — העקרון דומה לשרשרת המנוהלת עבור שטחי גלישה נפרדים. הרשומות הגולשות מאוחסנות במקום פנוי בשטח הראשי ומשורשרות אל המקום המקורי שלהן. כאשר רשומה חדשה מגיעה, והמקום תפוס על־ידי רשומה גולשת יש להזיז את הרשומה הגולשת ממקומה כדי לפנות מקום לרשומה המקורית ולעדכן את המצביעים בהתאם. התוכנה לניהול בסיסי נתונים TOTAL משתמשת בשיטה זו.

דוגמא:



תרשים 6-11: שרשור רשומות בשטח אחסנה ראשי

רשומות 750 ו־600 מאוחסנות בשטח הראשי ומשורשרות לרשומה 300. כאשר מגיעה רשומה שהכתובת המקורית שלה היא בכתובת התפוסה על־ידי הרשומה הגולשת 750, יש להזיז את רשומה 750 למקום אחר ולעדכן את המצביעים בהתאם.



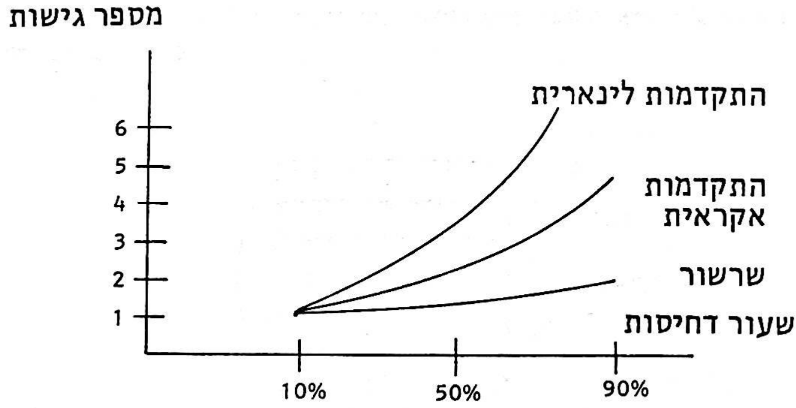
תרשים 6-12: הזזת רשומות גולשות

2. **מרחב פתוח** (Open Addressing) — בשיטה זו אין שימוש במצביעים. כאשר יש גלישה מתחיל תהליך של חיפוש קדימה בתוך הקובץ עד למציאת מקום פנוי ראשון. במקום זה מאחסנים את הרשומה הגולשת. לאיתור הרשומה ניגשים לכתובת המקורית. אם מצוין שהיתה גלישה בכתובת זו מתחיל חיפוש קדימה (לינארי) עד למציאת רשומה עם מפתח מתאים או עד למציאת מקום ריק ראשון. כאשר מגיעים לסוף הקובץ והרשומה לא נמצאה, ממשיכים את החיפוש מהתחלה, כלומר מתייחסים לקובץ כאל לולאה. לשיטה זו קוראים **התקדמות רציפה או לינארית** (Linear Probing).

בעת ביטול רשומה אין למחוק אותה באופן פיסי מאחר והמקום הפנוי יכול לגרום להפסקת תהליך החיפוש בצורה לא נכונה. יש לסמן את הרשומה המבוטלת ולמחוק אותה רק כאשר מבצעים אירגון מחדש של הקובץ.

השיטה הזאת דורשת זמן איתור רב יותר לעומת שיטת השרשור, אבל היא פשוטה יותר מבחינת העדכון מאחר ואין בה שימוש במצביעים. כאשר שיעור הדחיסות של הקובץ נמוך, זמן החיפוש של רשומה קצר יחסית.

כאשר שיטת ההתקדמות הרציפה יוצרת צפיפות של רשומות גולשות באיזורים מסוימים בשטח האחסנה ניתן להשתמש בשיטת **ההתקדמות האקראית** (Random Probing). בשיטה זו כאשר הכתובת המקורית תפוסה, מחשבים מספר אקראי ומחברים אותו לכתובת ליצירת כתובת חדשה. המער-כת בודקת אם הכתובת החדשה פנויה. במידה שלא, חוזרים על התהליך פעם נוספת, עד למציאת כתובת פנויה. שיטה זו נוטה לפזר טוב יותר את הרשומות הגולשות בקובץ. לאיתור הרשומה משתמשים באותה שיטה כדי ליצור את הכתובות מחדש. גם בשיטה זו אין לבטל רשומות אלא רק לסמן אותן כמבוטלות עד לארגון הקובץ מחדש.

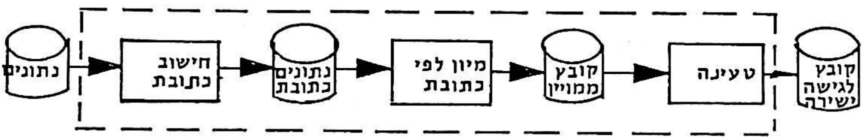


תרשים 6-13: מספר הגישות כפונקציה של שיעור דחיסות הקובץ

בתרשים 6-13 ניתנת השוואה גרפית של שיטות הגישה השונות. כפי שניתן לראות, שיטת השרשור מושפעת מעט מאד משיעור דחיסות הקובץ והיא נחשבת ליעילה ביותר. עבור שיעורי דחיסות נמוכים שלשת השיטות נותנות תוצאות כמעט זהות, ולכן נעדיף את שיטת ההתקדמות הלינארית בגלל פשטותה. ככל ששיעור הדחיסות עולה, שיטת השרשור עדיפה ואמנם, רוב המערכות לניהול קבצים משתמשות בשיטה זו.

6.8 טעינת קובץ לגישה ישירה

קיימות מספר שיטות לטעינת קובץ לדיסק. רוב המערכות לניהול נתונים מקצות את כל שטח האחסון הראשי מראש. אם נבנה את הקובץ בסדר אקראי התהליך יהיה מאד לא יעיל מאחר וכל פעם יכתב גוש אחר לדיסק. כדי לייעל את השיטה מבצעים סריקה של הקובץ, מחשבים את הכתובות וממיינים לפי הכתובת וטוענים את הקובץ ברצף. בשיטה זו כל הרשומות המיועדות לגוש אחד יכתבו בפעם אחת. טעינת הקובץ מבוצעת לרוב בעזרת תוכניות שירות המסופקת עם התוכנה.



תרשים 6-14: תהליך בנית קובץ לארגון ישור

בתרשים 6-14 מתואר תהליך **טעינה במעבר אחד** (One-Pass Load) באמצעות תוכנית שירות. קיימות שיטות בהן מתבצעת **טעינה בשני מעברים** (Two-Pass Load). בשלב ראשון טוענים לקובץ רק את הרשומות שלא גולשות. בשלב שני טוענים את כל הרשומות הגולשות. שיטה זו יכולה לייעל את תהליך הטעינה מאחר ואין צורך בהזזת רשומות גולשות שתפסו מקומות של רשומות מקוריות.

לסדר הופעת הרשומה בשרשרת יש חשיבות עבור זמן התגובה של עיבוד הקובץ. אם הרשומות היותר פעילות מופיעות בסוף השרשרת יש לבצע מספר גישות רב כדי להגיע אליהן ועל ידי כך גדל זמן העיבוד. קיימת אפשרות לדאוג שהרשומות היותר פעילות תופענה בשרשרת קרוב ככל האפשר לכתובת המקורית. לדוגמא, בקובץ מלאי נסדר את הרשומות לפי סדר יורד של תאריך תנועה אחרונה. סביר להניח שמספר השאילתות המתיחסות לפריטים שאין בהם תנועה יהיה קטן יותר מאשר לפריטים בהם יש תנועה רבה. שיטה זו קרויה **שרשור לפי פעילות** (Activity Rate Chaining) והיא מומלצת מאד כאשר ניתן לאפיין את שיעור הפעילות של הרשומות בקובץ.

6.9 סיכום

יתרונות

— אין צורך בגישה לאינדקסים ולכן גישה מהירה מאד.
— תהליך הפיכה מפתח לכתובת פשוט יחסית ולכן מהיר מאד.

חסרונות

— אם משתמשים בשיטת הייחוס על־ידי חישוב יש לפתח מנגנון לטיפול בגלישות. השיטה מותאמת לסוג המפתחות והתפלגות הערכים שלהם.
— דרושה הקצאת מראש של כל שטח האחסון הדרוש. יש לקבוע מראש את מספר הרשומות או הגושים שיהיו בקובץ לשם הקצאתם בדיסק.
— אין אפשרות לעיבוד סדרתי של הקובץ מאחר והרשומות מפוזרות בו בצורה אקראית.

6.10 נספח — הפיכת כתובת יחסית לכתובת מוחלטת בייחוס ישיר

נציג שיטה להפיכת המפתח, שהינו הכתובת היחסית של הרשומה, לכתובת מוחלטת בדיסק. כלומר שיטה לקבלת מספר הגליל, המסילה והגוש שבו נמצאת הרשומה. נשתמש בסימונים הבאים:

N — מספר הרשומות במסילה, נניח שכל רשומה תופסת גוש אחד.

M — מספר המסילות בגליל.

K — המפתח של הרשומה המבוקשת (שים לב — המפתח הראשון הוא אפס ולא 1).

התהליך החישובי מחולק לשני שלבים:

א. חלוקת המפתח של הרשומה במספר הרשומות במסילה

$$\frac{K}{N} = Q\ [R]$$

בעקבות החלוקה נקבל מנה Q ושארית R. השארית תבטא את המספר הסידורי של הרשומה במסילה.

ב. חלוקת המנה Q במספר המסילות לגליל.

$$\frac{Q}{M} = C\ [R_1]$$

בעקבות החלוקה נקבל מנה C ושארית R_1:

C — מספר הגליל

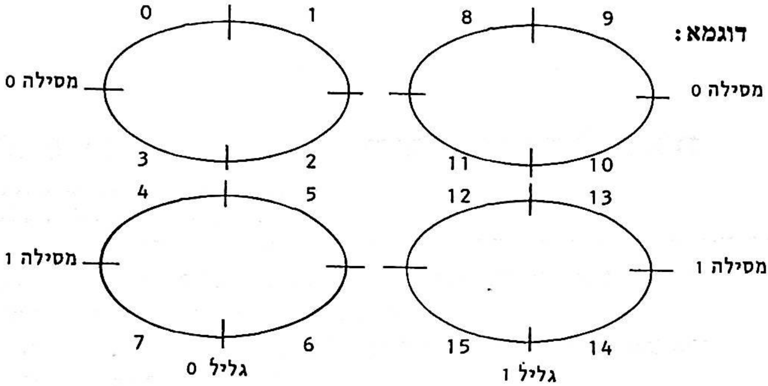
R_1 — מספר סידורי של המסילה בגליל.

תוצאת החישוב מפיקה כתובת יחסית בצורה של כתובת מוחלטת:

מספר סידורי	מספר מסילה	מספר גליל
R	R_1	C

כדי להפוך כתובת זו לכתובת מוחלטת יש להוסיף את המספר 1 לערכים אלו, כמפורט בדוגמא.

דוגמא:



$N = 4$ = מספר רשומות במסילה

$M = 2$ = מספר מסילות בגליל

$K = 7$ = מפתח הרשומה המבוקשת

<u>א.</u> $1\,[3] = \frac{7}{4}$

הרשומה המבוקשת נמצאת במקום הרביעי (1 + 3) במסילה.

<u>ב.</u> $0\,[1] = \frac{1}{2}$

הרשומה המבוקשת נמצאת בגליל הראשון (0) במסילה השניה (1 + 1).

הכתובת המתקבלת:	מספר סידורי	מספר מסילה	מספר גליל
כתובת יחסית:	3	1	0
כתובת מוחלטת:	4	2	1

פרק שביעי
אינדקסים משניים
(Secondary Index)

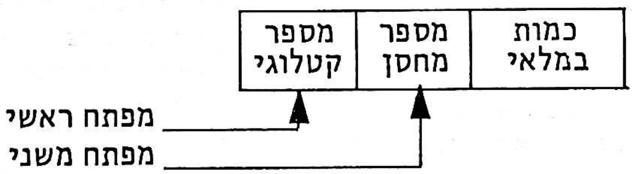
בפרק זה נציג את השיטה המאפשרת גישה לקבצים בעזרת מפתחות נוספים למפתח הראשי.

7.1 מבוא

שיטות ארגון הקבצים שהוצגו עד כה משתמשות **במפתח אחד בלבד** כמפתח חיפוש. במערכות המידע הממוחשבות המתקדמות התומכות באיחזור נתונים מהיר יש לבנות מנגנונים לאיחזור הרשומות לפי **מפתחות נוספים** שאינם מפתחות הסדר הקבוע של הקובץ. לדוגמא, במערכת לניהול כוח אדם יתכנו דרישות לגישה לרשומות לפי מספר עובד, מספר המחלקה של העובד או מספר הפרויקט בו הוא מועסק. השיטות שהוצגו עד עתה מאפשרות את הגישה לרשומות רק לפי מפתח אחד. בפרק זה נציג שיטה המאפשרת גישה לקבצים לפי מפתחות נוספים.

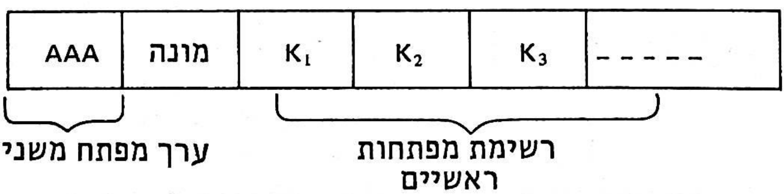
7.2 מפתח משני (Secondary Key)

לכל רשומה יש מפתח סדר אחד המשמש בדרך כלל גם כמפתח חיפוש. זהו **המפתח הראשי של הקובץ** (Primary Key) אשר משמש את המערכת לניהול קבצים לבניית אינדקס או לחישוב הכתובת של הרשומה. בנוסף למפתח הראשי ניתן להגדיר **מפתח משני אחד או יותר** למטרות שליפת הרשומות, מכיון שזהו תפקידם העיקרי.



תרשים 7-1: דוגמא למפתח ראשי ומשני בקובץ מלאי

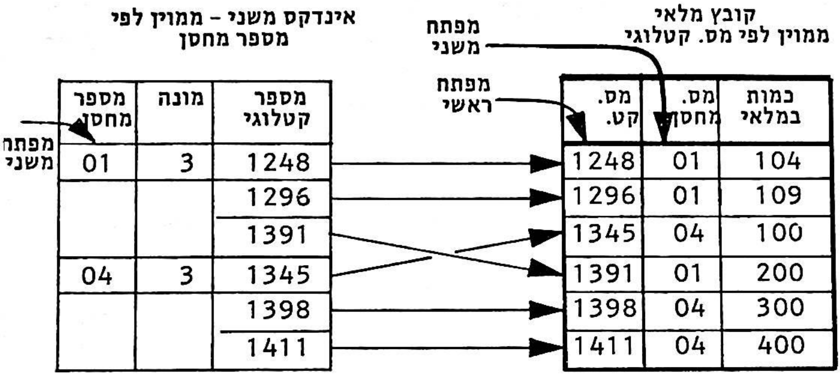
על מנת לאפשר גישה לרשומות לפי המפתח המשני יש לבנות אינדקס אחד נוסף. אינדקס זה יכיל כניסה אחת עבור כל ערך שהמפתח המשני קיבל, ואת רשימת כל המפתחות הראשיים המכילים ערך זה, כפי שמוצג בתרשים 7-2.



תרשים 7-2: מבנה רשומת אינדקס משני

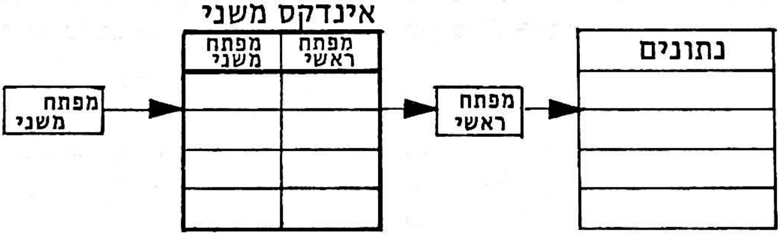
האינדקס המשני ממוין לפי המפתח המשני ולכן ניתן לבצע בו חיפושים גם לפי **מפתחות חלקיים** (Major Key) בדומה לחיפוש בשיטת הארגון האינדקס סדרתי. בגלל ההיפוך בין המפתח המשני והראשי באינדקס המשני נהוג גם לקרוא לאינדקס המשני **אינדקס הפוך** (Inverted File).

דוגמא: קובץ מלאי מכיל את הרשומות שבהן המפתח הראשי הוא מספר קטלוגי והמפתח המשני הוא מספר המחסן. לכל ערך שהמפתח המשני מקבל נבנה כניסה באינדקס המשני המכילה את רשימת המספרים הקטלוגיים בהם הערך הזה מופיע.



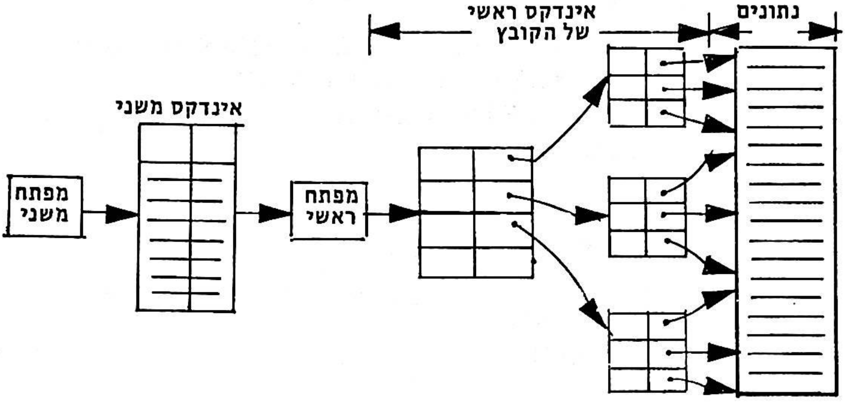
תרשים 7-3: מבנה קובץ עם שני מפתחות ומבנה האינדקס המשני

החיפוש באינדקס המשני מתבצע לפי מספר מחסן. כאשר מאתרים רשומה מבוקשת (מספר מחסן רצוי) מקבלים את רשימת כל הרשומות בקובץ בהן מופיע המחסן הזה. לדוגמא, הכניסה הראשונה באינדקס המשני מצביעה על שלש רשומות המכילות את הערך 01 בשדה "מספר מחסן". בעזרת המפתחות הראשיים ניתן לגשת לקובץ הנתונים ולשלוף את הרשומות, כפי שמוצג בתרשים 7-4.



תרשים 7-4: תרגום מפתח משני לראשי בעזרת האינדקס המשני

הגישה לקובץ הנתונים היא לפי המפתח הראשי בלבד, שהוא מפתח הסדר הרגיל של הקובץ. לכן קובץ הנתונים יכול להיות בארגון אינדקס סדרתי או בארגון לגישה ישירה. לדוגמא, אם קובץ הנתונים הוא קובץ אינדקס סדרתי נקבל את המודל המוצג בתרשים 7-5.



תרשים 7-5: אינדקס משני ואינדקס סדרתי

האינדקס המשני יכול להכיל מספר רמות אינדקס, בדומה לאינדקס הראשי, כדי ליעל זמן החיפוש בו. מתוך המודל הזה ניתן לראות שבניית האינדקס המשני אינה משפיעה על הנתונים המופיעים בקובץ הנתונים מאחר והגישה אליו מתבצעת בצורה הרגילה.

הגדרת מפתחות משניים ב-COBOL

בדוגמא שלפנינו קובץ הנתונים בנוי כקובץ אינדקס סדרתי עם מפתח ראשי: מספר קטלוגי (MISP-CATALOGI). בנוסף מוגדר המחסן (MAHSAN) במפתח משני עם ערכים חוזרים. הוראת ASSIGN במשפט SELECT מתיחסת לשני קבצים — לקובץ הנתונים ולקובץ האינדקס המשני. בתוך תוכנית קובול ניתן לגשת לרשומות הן לפי המפתח הראשי והן לפי המפתח המשני.

את המפתח המשני מגדירים במשפט SELECT בצורה דומה להגדרת המפתח הראשי.

```
SELECT MLAY-FILE ASSIGN TO XX    YY
          ORGANIZATION IS INDEXED
          RECORD KEY IS MISP-CATALOGI
          ALTERNATE RECORD KEY IS MAHSAN
                                  WITH DUPLICATES
                       ACCESS MODE IS DYNAMIC.
```

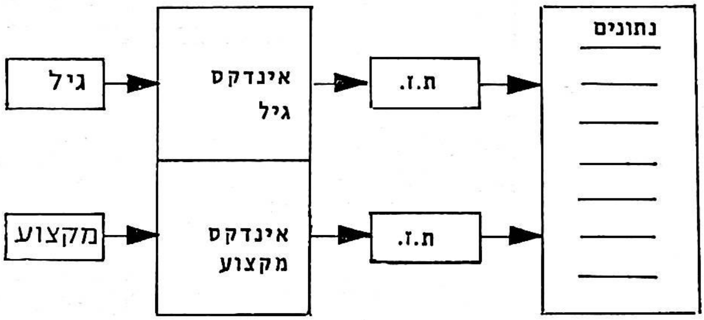
```
PROCEDURE DIVISION.
        ————
        OPEN INPUT MLAY-FILE.
        ————
        MOVE 04 TO MAHSAN.
        START MLAY-FILE KEY NOT LESS THAN MAHSAN
            INVALID KEY GO TO NOT-FOUND.
        READ MLAY-FILE INVALID KEY
            PERFORM ABNORMAL.
LOOP.
        PERFORM TIPUL.
        READ MLAY-FILE AT END
            PERFORM SOF.
        IF MAHSAN = 04
            GO TO LOOP.
```

בדוגמא זו יש פנייה לאינדקס המשני לרשומה עם מפתח מחסן 04. פקודות READ הבאות מביאות בזו אחר זו את כל הרשומות בהן מופיע המחסן 04. במקרה זה מפתח ההתיחסות (Key-of-Reference) הוא המפתח המשני ולכן המצביע מורה על האינדקס המשני. כאשר מבצעים פקודת קריאה בעזרת המפתח הראשי, מפתח ההתיחסות משתנה למפתח ראשי והמצביע משתנה בהתאם. שיטה זו מאפשרת גישות לקובץ לפי מפתח ראשי ומשני באותה תוכנית וקבלת הרשומה הבאה (READ NEXT) בהתאם.

בדוגמא הנ"ל התוכנית בודקת אם המפתח המשני השתנה כדי לדעת אם היא סיימה לקרוא את כל הרשומות עם מחסן 04. קיימות מערכות לניהול קבצים המודיעות באופן אוטומטי למשתמש שהערך של המפתח המשני התחלף.

7.3 מבנה ושימוש באינדקס משני

כאשר מגדירים לאותו קובץ מספר מפתחות משניים יש לכל אחד מהמפתחות אינדקס נפרד. באינדקס תהיה כניסה אחת עבור כל ערך שהמפתח המשני מקבל בקובץ. בדוגמא שבתרשים 7-6 מוצג קובץ נתונים המאורגן לפי מפתח ראשי של תעודת זהות (ת.ז.). לקובץ יש שני אינדקסים — אחד עבור גיל ואחד עבור מקצוע. אפשר להחזיק ברשומת האינדקס את מספר תעודת הזהות כפי שהוצג קודם. אך אם נחזיק בתוך האינדקס לא את המפתח של הרשומה (כמו תעודת זהות) אלא את הכתובת הפיסית הרי שהגישה מהאינדקס לנתונים תהיה מהירה ביותר. ניתן לעשות זאת אם קובץ הנתונים מאורגן בארגון ישיר עם ייחוס לפי טבלה.



תרשים 7-6: שני אינדקסים משנים לאותו קובץ

החיפוש באינדקס המשני יכול להתבצע תוך שימוש במספר רמות אינדקס בדומה לארגון אינדקס סדרתי או על־ידי ארגון קובץ האינדקס כקובץ בגישה ישירה וחישוב הכתובת.

הבעיה בניהול האינדקס המשני נובעת ממספר המופעים המשתנה של מפתחות ראשיים לכל אחד מהערכים האפשריים של המפתח המשני. קימות מספר אפשרויות לניהול האינדקס המשני.

א. שרשור רשומות בקובץ הנתונים. בשיטה זו כל הרשומות בעלות ערך זהה של מפתח משני משורשרות. באינדקס מופיעה רק כניסה אחת לכל ערך של המפתח המשני המצביעה על הרשומה הראשונה. ראה תרשים 7-7/1.

ב. שרשור רשומות תואם חומרה. בשיטה זו משורשרות כל הרשומות בעלות ערך זהה של מפתח משני המופיעות באזור חומרה אחד, למשל גליל בדיסק. באינדקס מופיעות מספר כניסות לאותו ערך של המפתח המשני, כמספר אזורי החומרה בהן נמצאות הרשומות. ראה תרשים 7-7/2.

מפתח משני
מפתח ראשי

קובץ נתונים

שרשור	כמות במלאי	מספר מחסן	מספר קטלוגי
	104	01	1248
	109	01	1296
	100	04	1345
	200	01	1391
	300	04	1398
	400	04	1411

אינדקס משני

מספר קטלוגי	מספר מחסן
1248	01
1345	04

7-7/1: שרשור רשומות הקובץ

קובץ נתונים

שרשור	כמות במלאי	מספר מחסן	מספר קטלוגי	
	104	01	1248	גליל 1
	109	01	1296	
	100	04	1345	
	200	01	1391	גליל 2
	300	04	1398	
	400	04	1411	

אינדקס משני תואם חומרה

מספר קטלוגי	מספר מחסן
1248	01
1391	01
1345	04
1398	04

7-7/2: שרשור רשומות תואם חומרה

קובץ נתונים

כמות במלאי	מספר מחסן	מספר קטלוגי
104	01	1248
109	01	1296
100	04	1345
200	01	1391
300	04	1398
400	04	1411

אינדקס משני

שרשור	מספר קטלוגי	מספר מחסן
	1248	01
	1345	04
	1296	
	1391	
	1398	
	1411	

7-7/3 : שרשור אינדקס משני

תרשים 7-7: מבנים שונים לניהול אינדקס משני

ג. שרשור רשומות האינדקס. בשיטה זו השרשור הוא של רשומות האינדקס ולא של רשומות הנתונים. ראה תרשים 7-7/3.

ד. מפות בינריות (Bit Map). לכל רשומה בקובץ יש באינדקס רצף של סיביות. כל סיבית מייצגת תכונה או ערך של שדה המשמש כמפתח משני. אם הרשומה מכילה ערך מסוים הסיבית תופעל (1) או שלא תופעל (0). שיטה זו נוחה מאד כאשר המפתח המשני יכול לקבל מספר מצומצם של ערכים בדידים. אם המפתח המשני יכול לקבל ערכים רציפים יש לבצע חלוקה של התחום.

מפתח ראשי	זכר	גיל עד 24	גיל 25-40	גיל 40 ומעלה
100	1	0	1	0
124	0	0	0	1
130	0	0	1	0
139	1	1	0	0

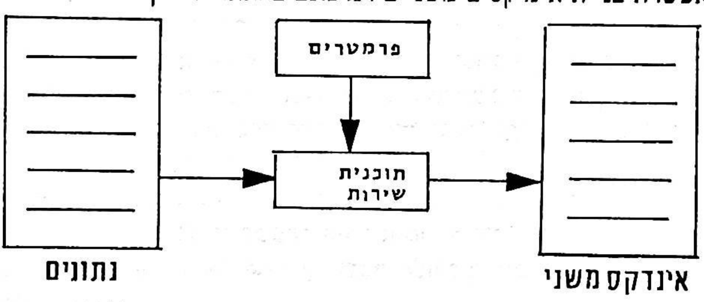
גיל	מין	מפתח ראשי
27	נקבה	130
36	זכר	100
45	נקבה	124
22	זכר	139

תרשים 7-8: שימוש במפה בינרית

בתרשים 7-8 מוצגת דוגמא שבה מבקשים לדעת את רשימת כל העובדים אשר גילם בין 25 ל־40. לשם כך נסרוק את העמודה המתאימה בתוך האינדקס וכאשר "נפגוש" בסיבית מופעלת נדע שהתנאי המבוקש מתמלא. נשלוף את המפתח הראשי .מטבלת האינדקס ונעבור לגישה לקובץ. מתוך הדוגמא המפורטת כאן אפשר להיווכח בעובדה שניתן בשיטה זו לסרוק במעבר אחד שני אינדקסים משניים או יותר, ואף להתנות אינדקס אחד באינדקס שני. לדוגמא, אפשר לחפש "זכר" וגיל "עד 24". בדוגמא הנ"ל הרשומה עם מפתח ראשי 139 ממלאת את התנאי.

כפי שניתן לראות השימוש באינדקס המשני אינו קשור כלל במבנה של קובץ הנתונים. אין בקובץ הנתונים מצביעים מיוחדים למטרה זו ולמעשה האינדקס המשני "שקוף" מבחינת קובץ הנתונים. בגלל עובדה זו ניתן בקלות רבה להוסיף מפתחות משניים או לבטל מפתחות משניים בהתאם לצורך מבלי לבצע שינויים כלשהם מקובץ הנתונים.

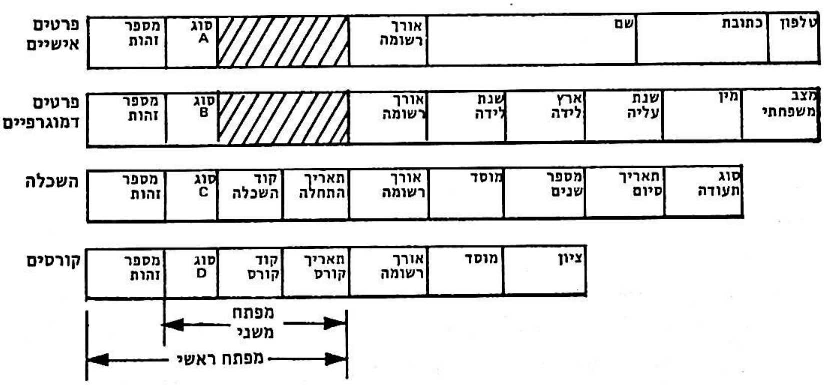
התוכנה לניהול נתונים מספקת בדרך כלל תוכנית שירות (Utility) המאפשרת בניית אינדקסים משניים ועדכונם בהתאם לצורך.



תרשים 7-9: בנית אינדקס משני בעזרת תוכנית שירות

בשיטות הארגון הנוספות שנדון בהן בהמשך הגישה לרשומות שלא לפי המפתח הראשי מחייבת שימוש במצביעים המופיעים בתוך רשומת הנתונים עצמה. על המשתמש לבחור בזהירות רבה את השדות בקובץ שישמשו כמפתח משני. מאחר והמערכת לניהול קבצים צריכה לעדכן באופן אוטומטי את האינדקס המשני, הרי שעדכון הקובץ הראשי גורמת גם לעדכון הקובץ המשני ולכן משך זמן העדכון ארוך יותר. יש לבנות אינדקס משני רק עבור מפתחות שידוע שיהיו בשימוש רב מאחר ובניית מספר רב של מפתחות משניים שאינם בשימוש מגדילה את זמן העדכון ללא כל צורך.

תרשים 7-10 מציג דוגמא של בניית מפתחות משניים החופפים בחלקם עם המפתחות הראשיים. נשתמש בדוגמא שהובאה בפרק הדן בתכנון רשומות.



תרשים 7-10: דוגמא לבנית מפתחות משניים

נבדוק את אפשרויות שליפת הנתונים מתוך קובץ המאורגן בצורה זו:

שליפת רשומת פרטים אישיים — נעביר למפתח מספר זהות וסוג רשומה A.

שליפת רשומת פרטים דמוגרפיים — נעביר למפתח מספר זהות וסוג רשומה B.

שליפת רשומת השכלה — אם דרושה רשומה מסוימת נעביר למפתח את כל הפרטים הרלוונטיים: מספר זהות וסוג C, קוד השכלה ותאריך.

— אם דרושות כל רשומות ההשכלה של עובד נתון נשתמש במפתח ראשי חלקי, מספר זהות וסוג רשומה C.

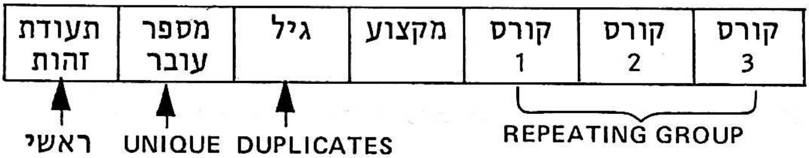
— אם דרושה רשימת כל העובדים בעלי השכלה בקוד מסויים, נשתמש במפתח המשני ונעביר למפתח סוג רשומה C וקוד השכלה.

שליפת רשומת קורס — התהליך דומה לתהליך השליפה של רשומת השכלה.

המפתחות המשניים יכולים להיות משני סוגים: **יחידים** (Unique), **כפולים** (Duplicates).

— מפתח משני יקרא יחיד כאשר לכל ערך שהוא מקבל קיימת רק רשומה אחת בקובץ הנתונים בה מופיע ערך זה.

— מפתח משני יקרא כפול אם הערך שלו מופיע במספר רשומות נתונים. את רשימת כל המפתחות הראשיים בעלי אותו ערך במפתח המשני ניתן לנהל בכמה אופנים: לפי סדר ההגעה של הרשומות או במיון לפי המפתח הראשי.



תרשים 7-11: סוגי מפתחות משניים

בתרשים 7-11 מוצגת רשומה בקובץ עובדים שבה המפתח הראשי הוא תעודת זהות. אם נגדיר את מספר העובד כמפתח משני הרי שקיימת התאמה חד־חד־ערכית בין מספר עובד ומספר תעודת זהות. לא יתכנו באותו מפעל שני עובדים להם אותו מספר עובד. אך אם נגדיר את הגיל כמפתח משני יתכנו מספר עובדים במפעל בעלי אותו גיל.

שדה מסוים ברשומה יכול לחזור מספר פעמים ברשומה ולקבל ערכים שונים, כמו למשל במקרה ששומרים הסטוריה של שלשת הקורסים האחרונים בהם השתתף העובד. זהו **שדה חוזר** (Repeating Group). כאשר נבנה על השדה הזה אינדקס, יהיו מספר רשומות אינדקס אשר יצביעו על אותה רשומת נתונים כפי שמתואר בתרשים 7-12.

נתונים

	קורס 3	קורס 2	קורס 1	עובד
	C	B	A	10
	D	C	A	20
	E	C	B	40
	D	C	A	50

אינדקס משני

A	10, 20, 50
B	10, 40,
C	10, 20, 40, 50
D	20, 50
E	40

תרשים 7-12: אינדקס משני עם שדה חוזר

השימוש באינדקסים משניים מאפשר ביצוע **חיפוש לפי מספר מפתחות** (Multiple Key Search) בו־זמנית מבלי שיהיה צורך לגשת לנתונים עצמם.

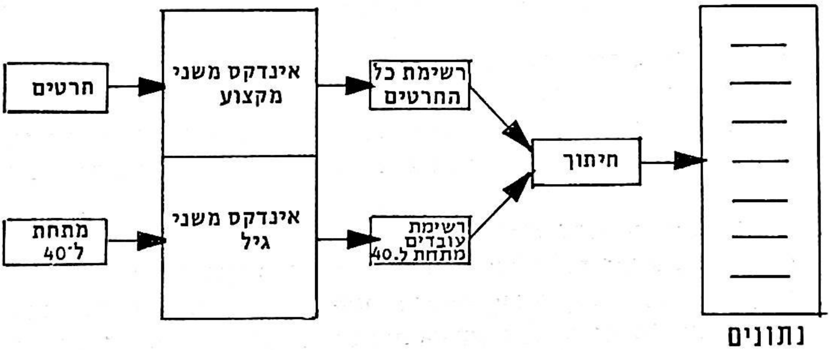
נשתמש בדוגמא הקודמת של קובץ עובדים ונציג את השאילתה הבאה:

"רשימת כל החרטים מתחת לגיל 40".

ניתן לגשת לאינדקס משני "מקצוע" ולשלוף את כל תעודות הזהות של החרטים. לאחר מכן לגשת לאינדקס משני "גיל" ולשלוף את תעודות הזהות של כל העובדים במפעל שהם מתחת לגיל 40. רק תעודות הזהות המופיעות בשתי הרשימות גם יחד (חיתוך) עונות על השאילתה, כמוצג בתרשים 7-13.

זו דוגמא לשאילתה שהתשובה עליה התקבלה מבלי שיהיה צורך לגשת לנתונים לבצע בהם סריקות נרחבות. קיימות מספר מערכות לניהול נתונים המאפשרות ביצוע חיתוכים כאלה בצורה יעילה מאד. לדוגמא מערכת לניהול בסיסי נתונים ADABAS. מערכות אלה מאפשרות שאילתות מורכבות כגון:

"רשימת כל החרטים או רתכים שהם מתחת לגיל 40 ומעל 28 ומקום מגוריהם בפתח־תקוה".



תרשים 7-13: חיתוך בין שני מפתחות משניים

7.4 סיכום

יתרונות:

— אפשרות גישה לרשומה לפי מספר רב של מפתחות. מאחר והמפתחות המשניים ממויינים באינדקס ניתן לבצע חיפוש גם לפי מפתח חלקי.
— אפשרות ביצוע חיתוך בין מפתחות משניים מבלי שיהיה צורך לגשת לנתונים עצמם.
— ניתן לקרוא את הקובץ גם ללא שימוש באינדקס משני.
— השימוש באינדקס משני אינו גורם לכל תוספות בקובץ הנתונים עצמו. מסיבה זו ניתן להוסיף או לבטל מפתחות משניים בצורה חיצונית לקובץ הנתונים.

חסרונות

— הטיפול באינדקס המשני יקר. כל עדכון בקובץ חייב להתבטא מיד באינדקס המשני. זמן העדכון הכולל של הקובץ עולה.
— הגישה לרשומה באמצעות אינדקס משני מחייבת ביצוע מספר גישות נוספות לדיסק לשם שליפת המפתח הראשי מתוך האינדקס.
— האינדקס עצמו תופס מקום אחסנה.

7.5 שאלות

1. מערכת מידע כוללת קובץ עובדים. יש למצוא את:
— כל העובדים הגרים בירושלים.
— כל העובדים שיש להם מעל שני ילדים הגרים בירושלים.
— כל העובדים בעלי משכורת של 10,000 שקל ומעלה ולהם יותר משני ילדים.

הסבר אילו מפתחות משניים יש לבנות לקובץ. כתוב את השאילתות בתוכניות קובול.

2. הסבר כיצד ניתן לענות לשאלה 1 ללא שימוש באינדקס משני.
3. מהם השיקולים לקביעת מפתחות משניים. מדוע לא לקבוע את כל השדות ברשומה, כמפתחות משניים.
4. קובץ רופאים מכיל את הנתונים הבאים:

בית חולים	התמחות	מספר רופא
תל־השומר	כירורגית	104
גהה	פסיכיאטריה	108
תל־השומר	ניתוחי לב	110
רמב"ם — חיפה	פנימית	117
רמב"ם — חיפה	פנימית	124
תל־השומר	כירורגית	131
רמב"ם — חיפה	עיניים	141
תל־השומר	פנימית	191

א. הצג את תוכן האינדקס המשני עבור "התמחות" ו־"בית־חולים".
ב. האם תוכנו של האינדקס ישתנה אם הקובץ יהיה בארגון אינדקס סדרתי או בארגון ישיר?

פרק שמיני
ארגון נתונים מורכב — שרשרות
(Complex Data Structures-Chains)

בפרק זה נציג מבנים ליניאריים של נתונים המשתמשים בשיטות שונות של שרשור רשומות.

8.1 מבוא

בשיטות הארגון שהוצגו בפרקים קודמים קיימות מספר מגבלות אשר הביאו לחיפוש דרכים לעקיפתן:

א. קיים קשר הדוק בין המפתח הלוגי לבין המיקום הפיסי של הרשומה בקובץ.

ב. המפתח מאפשר איתור של רשומה בודדת. השיטות לא נותנות תשובה למקרה שבו יש לאתר אוסף של רשומות (לדוגמא, כל העובדים השייכים למחלקה מסוימת).

ג. הטיפול בהוספת רשומות חדשות מסורבל מאד. ברוב השיטות קיימים מנגנונים לטיפול בגלישות, דבר המוריד את הביצועים. בחלק מהשיטות יש להגדיר את השטח מראש הרשומות החדשות.

בפרק זה נציג מספר שיטות לארגון נתונים בהם קיימת אי־תלות בין המפתח הלוגי של הרשומה לבין כתובתה הפיסית בדיסק. השיטות האלה משתמשות במנגנון של **מצביעים** (Pointers) להשגת אי־התלות. אי־התלות מונעת את הצורך במנגנונים מיוחדים לטיפול בגלישות ובצורך להגדיר מראש שטחים לגלישות. יש לציין שהשיטה של אינדקסים משניים מאפשרת השגת אי־תלות בין המפתח המשני והמיקום של הרשומה, אבל מאחר והמפתח המשני מתורגם למפתח ראשי מתקבלת התלות בצורה עקיפה. האינדקסים המשניים מאפשרים גם שליפת אוסף רשומות בעלי תכונה זהה. הבעיה בשיטת האינדקס המשני היא זמן העדכון הגדול הדרוש לטיפול באינדקסים.

מבחינים בין שלשה סוגי מצביעים:

— מצביע המכיל **כתובת מוחלטת** (Absolute Address) של הרשומה ביחידת האיחסון. שיטה זו מאפשרת את הגישה המהירה ביותר לרשומה המבוקשת.

— מצביע המכיל **כתובת יחסית** (Relative Address) של הרשומה ביחידת האיחסון. הכתובת היחסית אינה תלויה בכתובת המוחלטת של הרשומה. שיטה זו מאד נוחה מאחר וניתן להזיז את הקובץ ביחידת האיחסון ללא שינוי בתוכן המצביעים השונים. הקובץ גם לא חייב להיות מאוחסן ברצף אחד. הגישה איטית יותר יחסית לכתובת המוחלטת מאחר ויש לבצע תהליך תרגום בין הכתובת היחסית לכתובת המוחלטת. רוב המערכות משתמשות בשיטה זו.

— מצביע המכיל **מפתח של הרשומה** (Symbolic Pointer). על מנת להגיע לרשומה אשר המפתח שלה מופיע במצביע, יש לבצע את התהליך הרגיל של תרגום בין מפתח לכתובת, אם ע״י אינדקסים ואם ע״י תהליך חישובי

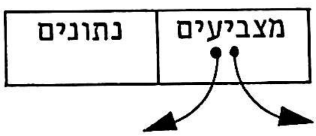
כלשהו. שיטה זו מאפשרת את הגמישות הרבה ביותר מאחר וניתן להזיז גם רשומה בודדות ממקומה מבלי שתוכן המצביע ישתנה. מאידך זוהי השיטה האיטית ביותר בגלל הצורך בתרגום מפתח־כתובת.

מבני הנתונים המשתמשים במצביעים לניהול המיקום והקשרים בין הרשומות הינם אבני הבנין של רוב המערכות המתקדמות לניהול בסיסי נתונים (כגון DL/1, IDMS, TOTAL) ומכאן החשיבות הרבה בהבנתם. מערכות ההפעלה משתמשות גם כן במבני שרשרות לניהול תורים של משימות המוטלות עליהן, או של אירועים שיש לפקח עליהם.

8.2 מבני נתונים משורשרים
(Chained/List Data Structure)

השיטות שהוצגו בפרקים קודמים מאפשרות שליפת רשומה בודדת. כאשר קיים קשר כלשהו בין הרשומות (שייכות לאותו נושא או לאותו פרט באוכלוסיה) ניתן לייצג אותו רק על־ידי הקירבה הפיסית ביניהן. דוגמא לכך ניתן למצוא בשיטת ארגון האינדקס הסדרתי שבו הרשומות ממוינות באופן פיסי לפי המפתח. הבעיה מתעוררת כאשר רשומה אחת קשורה במספר קשרים שונים לרשומות שונות. במקרה זה שיטת הקירבה הפיסית מחייבת החזקת הרשומה מספר פעמים בקובץ, דבר שאינו אפשרי בקבצים גדולים. השיטה לטיפול בבעיות אלה היא השימוש במצביעים.

המבנה האופיני של רשומה במבנה נתונים המשתמש במצביעים מופיע בתרשים 8-1. המצביע הינו שדה המכיל כתובת של הרשומה הבאה בקשר בין הרשומות. מאחר ורשומה אחת יכולה להשתתף במספר רב של קשרים היא עשויה להכיל מספר כלשהו של מצביעים. הרשומה יכולה להכיל גם שדה מפתח ראשי כלשהו, המשמש לאיתור מהיר שלה.



תרשים 8-1: מבנה רשומה במבנה נתונים עם מצביעים

אחד היתרונות של מבנה נתונים משורשר הוא יכולתו לטפל בקלות יחסית באינפורמציה בעלת אופי דינמי. לדוגמא ביישום של ניהול רכש יש לשמור אינפורמציה על הלקוח וכל ההזמנות שלו. מאחר ומספר ההזמנות ללקוח משתנה באופן קיצוני בין לקוחות והן כפונקציה של הזמן, יהיה זה בזבוני מאד לנהל אינפורמציה כזאת ברשומות בעלות אורך קבוע. שיטה יעילה יותר היא לקשור את רשומת הלקוח אל רשומות ההזמנה וליצור מבנה משורשר. זוהי דוגמא לשרשרת המכילה שני סוגי רשומות, רשומות לקוח ורשומות הזמנה. רוב המערכות לניהול בסיסי נתונים משתמשות במבנה זה לקישור בין סוגי רשומות.

החסרון העיקרי של מבני נתונים משורשרים נובע מהעובדה שסריקת שרשרת דומה לסריקת קובץ סדרתי. מספר ההשוואות הממוצע בשרשרת בעלת N רשומות והסתברות חיפוש שווה תהיה כמו בקובץ סדרתי.

$$\sum_{i=1}^{n} i\,P_i = \sum_{i=1}^{n} i\frac{1}{N} = \frac{N+1}{2}$$

בגלל הגמישות הרבה של השיטה, קיימים מספר רב מאד של מבנים המשתמשים במצביעים. נציג בהמשך את המבנה הפשוט ביותר — **שרשרת** — וכיצד ניתן לבנות ממנו את כל המבנים המורכבים.

נשתמש בסימונים הבאים:

K_i — מפתח לוגי של רשומה i

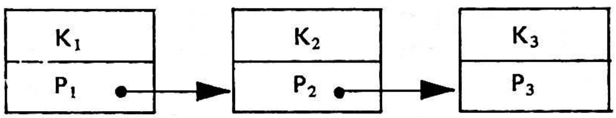
A_i — הכתובת הפיסית של רשומה i

P_i — תוכן המצביע ברשומה i. אם הוא מצביע על רשומה j למשל, נרשום

$$P_i = A(K_j)$$

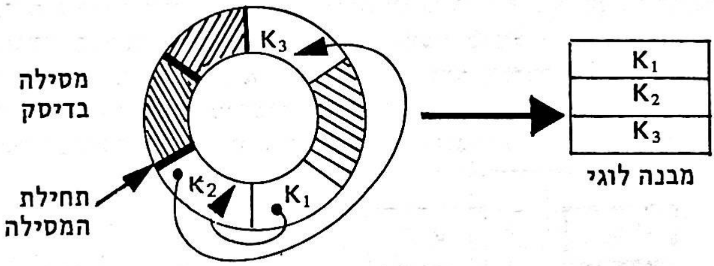
8.3 שרשרת פשוטה (Simple Chain)

שרשרת פשוטה מוגדרת כאוסף רשומות הקשורות ביניהן בעזרת מצביעים. סריקת השרשרת מתבצעת על ידי קריאת רשומה ושימוש במצביע שלה לקריאת הרשומה הבאה.



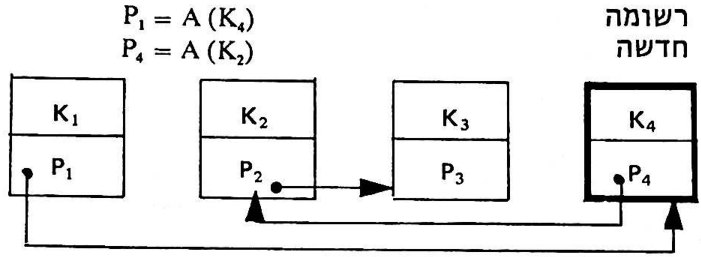
תרשים 8-2: שרשרת פשוטה

יש לעשות הבחנה בין המיקום הפיסי של הרשומה בקובץ לבין מקומה הלוגי בשרשרת. בתרשים 8-3 מוצגות מספר רשומות במסילה בדיסק. רשומה K_3 נמצאת לפני רשומה K_2 מבחינה פיסית אבל בגלל השימוש במצביעים רשומה K_2 מקדימה את K_3 מבחינה לוגית.



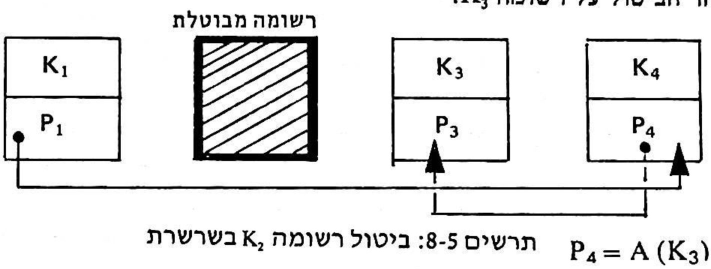
תרשים 8-3: מבנה פיסי ולוגי של שרשרת

הוספת רשומה חדשה לקובץ יכולה להתבצע במקום פיסי פנוי כלשהו בעוד הקישור הלוגי שלה מתבצע בעזרת המצביעים. לדוגמא, אם יש להוסיף רשומה חדשה עם מפתח K_4 המקיים את התנאי $K_1<K_4<K_2$ מקומה הלוגי של הרשומה הוא בין רשומות K_1 ו־K_2. בשיטות הרגילות יש צורך להזיז את רשומה K_2 ו־K_3 כדי לפנות מקום לרשומה החדשה. השימוש במצביעים מאפשר הוספת הרשומה למקום פנוי בסוף הקובץ. עדכון המצביעים ישמור על הסדר הלוגי של הרשומות כמוצג בתרשים 8-4.



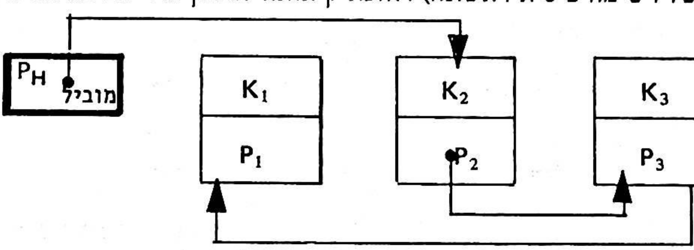
תרשים 8-4: הוספת רשומה חדשה K_4 לשרשרת

סריקת השרשרת, תוך שימוש במצביעים קובעת את הסדר הלוגי K_1, K_4, K_2, K_3. ביטול רשומה מהקובץ מתבצע גם־כן על־ידי עדכון המצביעים. תרשים 8-5 מדגים ביטול רשומה שהמפתח שלה K_2. המצביע P_4 מצביע לאחר הביטול על רשומה K_3.



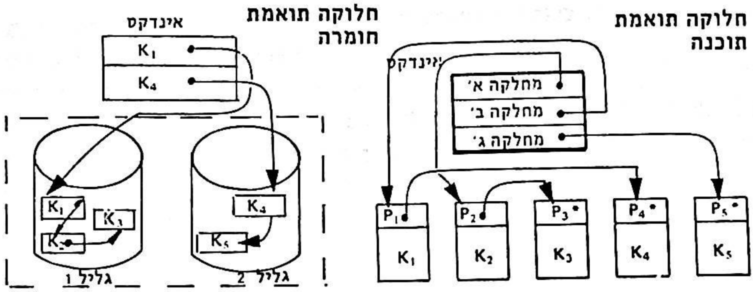
תרשים 8-5: ביטול רשומה K_2 בשרשרת

על מנת להתחיל את סריקת השרשרת יש לדעת את הכתובת של הרשומה הראשונה בשרשרת ולשם כך דרושה **רשומת מוביל** (Chain Header). בתרשים 8-6 הרשומה הראשונה בשרשרת היא רשומה K_2, לכן רשומת המוביל מצביעה עליה. את רשומות המוביל ניתן לשמור במקום מיוחד בקובץ (למשל רשימה פיסית ראשונה) ולהעתיק אותה לזכרון מיד עם פתיחת הקובץ.



תרשים 8-6: רשומת מוביל המצביעה על תחילת השרשרת

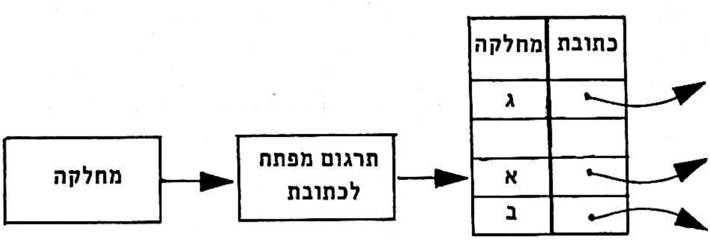
הרשומות בקובץ אינן חייבות להיות קשורות בשרשרת אחת בודדת ואפשר ליצור מספר שרשרות לנושאים שונים. לדוגמא, ניתן לשרשר את כל העובדים במחלקה. במקרה זה נקבל מספר שרשרות כמספר המחלקות במפעל, ולכל אחת מהן תהיה נקודת התחלה. מקובל לכנות מבנה זה — Multiway Access Chain באשר לפנינו מספר נקודות התחלה או גישה לקובץ אחד.



תרשים 8-7: מבנה שרשרת עם מספר נקודות כניסה

ניתן לחלק את השרשרת לתת־שרשרות הן משיקולי תוכנה והן משיקולי חומרה. חלוקת השרשרת לתת־שרשרות משיקולי חומרה נובעת מהצורך ליעל את תהליך החיפוש ומקובל לקרוא לשרשרת כזו, שרשרת תואמת חומרה (Cellular List, Hardware Zoned List). ניתן לבנות תת־שרשרת כך שכל הרשומות בה תופענה באותו אזור חומרה, גליל בדיסק למשל: סריקת תת־ש־רשרת כזאת אינה מחייבת הזזת ראשי קריאה/כתיבה של הדיסק. לכל תת־שרשרת יש נקודת כניסה נפרדת באינדקס מיוחד המכיל מצביע לרשומה הראשונה בכל אזור חומרה. דוגמא לשרשרת תואמת חומרה מופיע תרשים 8-7.

לשליפת רשומות העובדים השייכים למחלקה מסוימת יש לאתר תחילה את רשומת המוביל של המחלקה. אחת האפשרויות לניהול רשומות המוביל היא שמירתם בקובץ נפרד לגישה ישירה או לגישה בעזרת אינדקס.

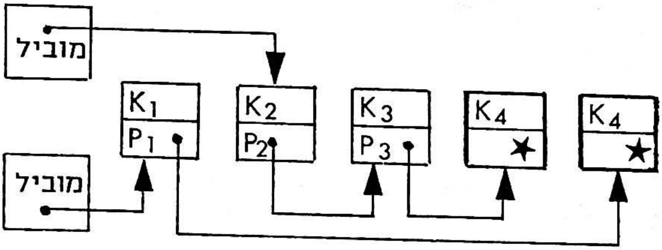


תרשים 8-8: גישה ישירה לרשומות מוביל

ניתן להחזיק ברשומות המוביל מידע נוסף כגון שם המחלקה ועוד. מקובל לשמור ברשומת המוביל גם את מספר הרשומות בשרשרת. מונה זה יאפשר מתן תשובות מהירות לשאילתות מסוימות ללא סריקת הקובץ (למשל, "מה מספר העובדים במחלקה ב'?"). בנוסף משמש המונה לקבלת החלטה לשרשרת החיפוש כאשר יש לסרוק מספר שרשרות. נושא זה יובהר בהמשך.

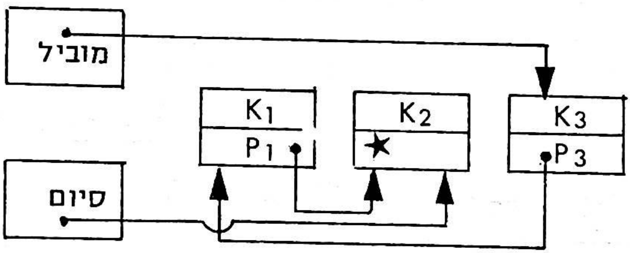
בנוסף לרשומת מוביל המאפשרת גישה לרשומה הראשונה בשרשרת, יש לדעת מיהי הרשומה האחרונה בשרשרת. קיימות מספר אפשרויות לציון נקודות הסיום של השרשרת.

א. **סימון מיוחד:** בשיטה זו נשתמש בסימן מיוחד (כוכבית למשל) לסימון הרשומה האחרונה. כאשר סורקים את השרשרת ומתקבל מצביע אשר מכיל את הערך המיוחד ($P_i = \star$) פרוש הדבר שהגענו לסוף השרשרת.



תרשים 8-9: סימון סיום שרשרת

ב. **הצבעה לרשומה אחרונה:** בנוסף לרשומת מוביל נשמור רשומה נוספת המצביעה על הרשומה האחרונה בשרשרת. פתרון אחר הוא לשמור מצביע נוסף בתוך רשומת המוביל, אשר מורה על הרשומה האחרונה בשרשרת. שיטה זו מאפשרת גישה מידית לסוף השרשרת ללא סריקת השרשרת. מצביע זה מייעל מאד את תהליך הוספת רשומות חדשות לקובץ מאחר ויש לעדכן מצביע זה בלבד.

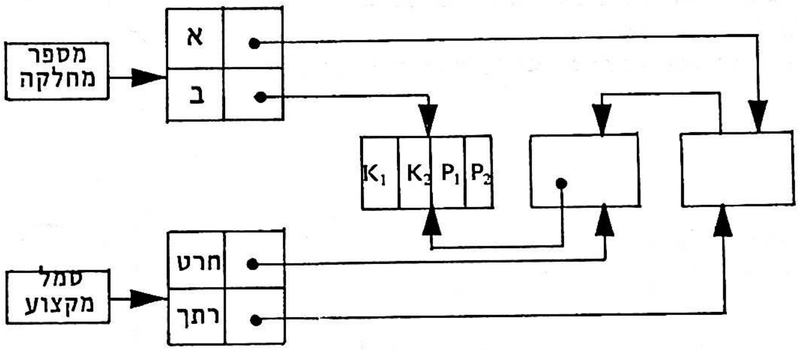


תרשים 8-10: מבנה שרשרת עם רשומת מוביל ורשומת סיום

שיתוף רשומה במספר שרשרות (Multilist)

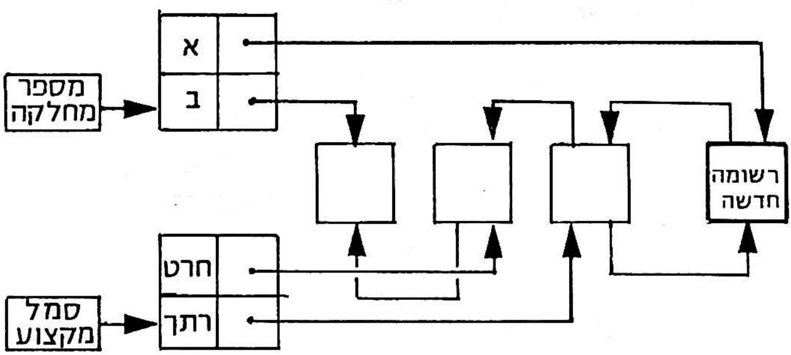
אפשר לשתף רשומה אחת מסוימת במספר שרשרות ונשתמש לשם כך בדוגמא של קובץ עובדים המוצגת בתרשים 8-11. נשרשר את כל רשומות העובדים במחלקה מסוימת ואת רשומות כל העובדים במקצוע מסוים. לשם הפשטות נניח כי במפעל רק שתי מחלקות ושני מקצועות ולכן תהינה ארבע נקודות התחלה. ניתן לראות שאותה רשומה משתתפת בשתי שרשרות שונות.

עבור כל שרשרת הרשומה מכילה את מפתח הסדר שלה ואת המצביע לרשומה הבאה, ולכן יהיו לכל רשומה שני מפתחות ושני מצביעים.



תרשים 8-11: שיתוף רשומה בשתי שרשרות

הוספת רשומה חדשה מחייבת עדכון המצביעים בשתי השרשרות בהתאם למקום הלוגי שלה בכל אחת מהן. נניח ששרשרת המחלקה ממוינת לפי תעודת הזהות של העובד ושרשרת המקצוע ממוינת לפי הדרגה המקצועית. בהוספת רשומה חדשה יש לקבוע מהו מקומה בכל אחת מהשרשרות. לדוגמא נוסיף רשומה חדשה אשר צריכה להיות ראשונה בשרשרת המחלקה (ראשונה במחלקה א׳) ואחרונה בשרשרת מקצוע (כלומר אחרונה ברשימת ״רתך״). ביטול רשומה משרשרת אחת אינו מאפשר את ביטולה הפיסי מאחר והיא עדיין שייכת לשרשרת השניה. לכן יש רק לסמן את הרשומות המבוטלות. רק רשומה מבוטלת מכל השרשרות בהן היא משתתפת ניתנת לביטול פיסי.

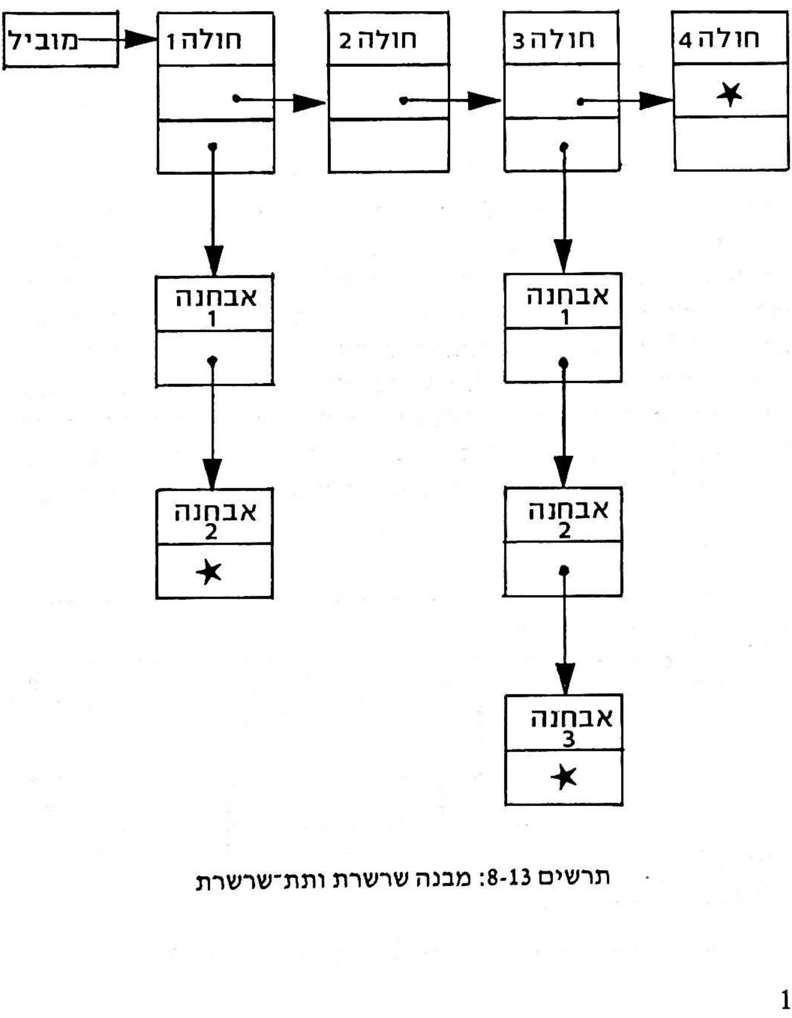


תרשים 8-12: הוספת רשומה המשתתפת בשתי שרשרות

תת־שרשרת (Sub-Chain)

קיימת אפשרות להצביע מתוך רשומה בשרשרת אחת אל רשומות בתת-־שרשרת אחרת. נשתמש בדוגמא שבתרשים 8-13: השרשרת הראשית מכילה את רשומות כל החולים המאושפזים בבית החולים. לכל רשומת חולה נבנה תת־שרשרת המכילה את כל רשומות האבחנה של החולה.

בדוגמא זו לכל רשומת חולה יש שני מצביעים: מצביע אחד אל הרשומה הבאה בשרשרת החולים, ומצביע שני אל הרשומה הראשונה בשרשרת רשומות האבחנה של החולה. ברשומות האבחנה יש רק מצביע אחד המצביע על רשומת האבחנה הבאה של אותו חולה.



תרשים 8-13: מבנה שרשרת ותת־שרשרת

8.4 מבני שרשרות מיוחדים

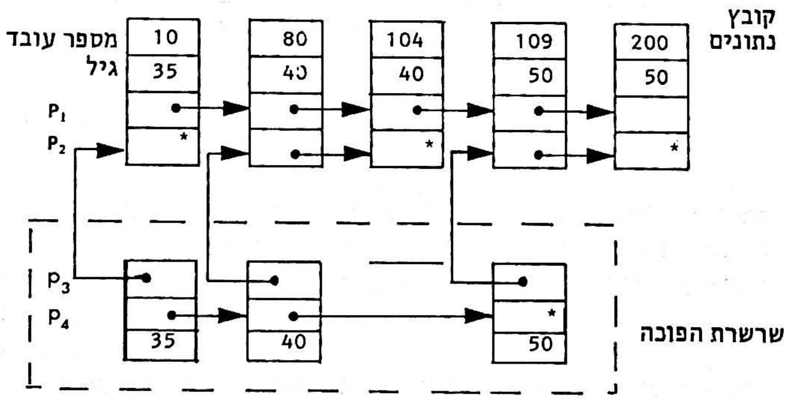
קיימים מספר סוגים של שרשרות המאופיינות על ידי מבנה או תכונה מיוחדת. בהמשך נסקור כמה מהם.

שרשרת הפוכה (Inverted List)

מבנה השרשרת ההפוכה דומה בעקרון לאינדקס המשני. במקום שימוש באינדקס המכיל כניסה עבור כל הרשומות המכילות ערך מסוים של המפתח המשני, משתמשים כאן בשרשור כל הרשומות האלה ובאינדקס המצביע לרשומה הראשונה בשרשרת. שיטה זו מפשטת את בעית ניהול האינדקס המשני שבו כל כניסה באינדקס מצביעה על מספר משתנה של רשומות בקובץ. במבנה זה האינדקס מצביע על הרשומה הראשונה בלבד.

דוגמא:

— קובץ הנתונים הינו במבנה שרשרת פשוטה חד־כיוונית.
— קובץ האינדקס ההפוך הינו במבנה שרשרת פשוטה חד־כיוונית.
— רשומות קובץ הנתונים מכילות בין השאר את השדות מספר עובד והגיל שלו. האינדקס ההפוך הוא על שדה הגיל.

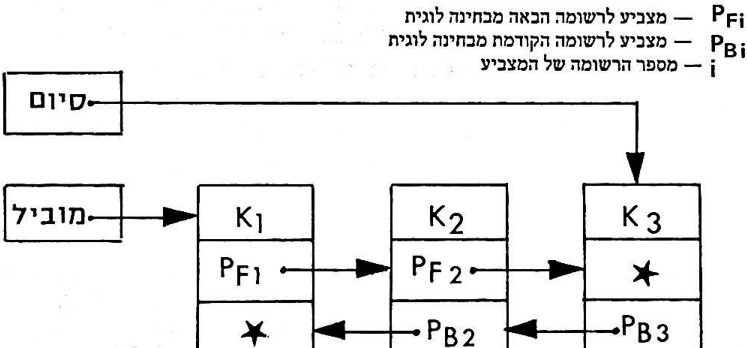


תרשים 8-14: שרשרת הפוכה

מתוך התרשים אפשר לראות שבקובץ הנתונים שני מצביעים P_1 ו־P_2. P_1 מצביע על הרשומה הבאה מבחינה לוגית לפי מספר עובד. P_2 מצביע על הרשומה הבאה שהעובד שלה באותו גיל. בקובץ האינדקסים שני מצביעים P_3 ו־P_4. P_3 מצביע על הרשומה הראשונה בשרשרת הגיל. P_4 מצביע על הרשומה הבאה מבחינה לוגית לפי גיל העובד.

שרשרת דו־כיוונית (Bi-Directional Chain)

בדוגמאות הקודמות הוצגו שרשרות חד־כיווניות בלבד. בשרשרות אלה, תהליך הוספה או ביטול רשומה הינו מורכב. על התוכנית לקרוא את השרשרות קדימה ול"זכור" את הכתובת הקודמת על מנת לאפשר את עדכון המצביעים. אם ניתן להגיע לרשומה בשרשרת בצורה ישירה כלשהי, אין אפשרות לדעת מיהי הרשומה הקודמת לה. בעיה נוספת היא שבמקרה של תקלה באחד המצביעים אין אפשרות לשחזר את השרשרת. על מנת להתגבר על בעיות אלה ניתן להשתמש בשרשרת דו־כיוונית כפי שמוצג בתרשים 8-15. במבנה זה יש בכל רשומה שני מצביעים — אחד לרשומה הבאה מבחינה לוגית ואחד לרשומה הקודמת מבחינה לוגית. שני המצביעים מאפשרים סריקת השרשרת בשני הכיוונים, לפנים ולאחור.



תרשים 8-15: מבנה שרשרת דו כיוונית

קיימים מצבים בהם סריקה לאחור חוסכת מיון. לדוגמא נניח שהשרשרת מכילה את כל האבחנות של חולה מסוים לפי תאריך האבחנה. אם נרצה את שלש האבחנות האחרונות של החולה ניתן להתחיל בסריקת השרשרת מהסוף (אם קיים מצביע לסוף) ולשלוף את שלש הרשומות הראשונות (כלומר האחרונות בשרשרת).

שני המצביעים מגדילים את אמינות המבנה מאחר ותקלה במצביע אחד מאפשרת את שחזורו על־ידי סריקת השרשרת מכיוון ההפוך. יש לזכור עם זאת ששימוש בשני מצביעים תופס מקום רב יותר בתוך הרשומה ותהליך העדכון מורכב יותר מאחר ויש לטפל בשלש רשומות בעת העדכון.

נדגים כאן אלגוריתם פשוט להוספת רשומה חדשה לשרשרת דו־כיוונית. האלגוריתם אינו מטפל במקרה שהרשומה החדשה היא בהתחלת השרשרת או בסופה. אך ניתן להרחיב אותו גם למקרים אלה.

הגדרות:

A_i כתובת רשומה i

P_{Fi} מצביע קדימה ברשומה i

P_{Bi} מצביע אחורה ברשומה i

A_i	K_i	$P_{F,i}$	P_{Bi}	A_{i+1}	K_{i+1}	$P_{F,\ i+1}$	$P_{B,\ i+1}$

$P_{F,i} = A(K_{i+1})$ $\qquad$ $P_{B,\ i+1} = A(K_i)$

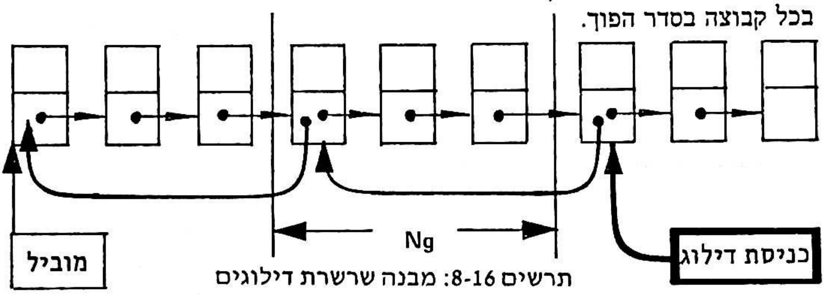
$P_{B,i} = A\ (K_{i-1})$ $\qquad$ $P_{F,\ i+1} = A(K_{i+1})$

נוסיף לקובץ רשומה j אשר כתובתה תהיה A_j. מבחינה לוגית רשומה זו נמצאת בין רשומה i לרשומה 1 + i. תהליך הוספת הרשומה ועדכון המצביעים מוצג בשלבי הוראות לביצוע:

1. אתר מקום פנוי הבא מתוך רשימת מקומות פנויים. המקום יסומן ב־Aj. עדכן רשימת מקומות פנויים.
2. קרא רשומות בשרשרת לפי מצביע קדימה. המשך לקרוא עד אשר המפתח של הרשומה שקראת גדול ממפתח הרשומה החדשה $(K_{i+1} > K_j)$
3. עדכן מצביע אחורה ברשומה 1 + i. (temp1 — כתובת זמנית עבור מצביע לאחור).
   $P_{B,i+1} = A\ (Kj)\ ; P_{B,i+1} = templ = A(Ki)$.
   כתוב רשומה 1 + i חזרה במקומה.
4. קרא רשומה שכתובתה A_i. בצע עדכון מביעים קדימה. (temp 2 — כתובת זמנית למצביע קדימה).
   $P_{F,i} = A(Kj)\quad ; P_{F,i} = temp\ 2$
   כתוב רשומה i חזרה במקומה.
5. עדכן מצביעים ברשומה החדשה.
   $P_{B,j} = temp\ 2; P_{F,j} = temp\ 2$
   כתוב רשומה j במקום Aj.
6. סוף תהליך הוספת רשומה חדשה.

שרשרת דילוגים (Skip-Chain)

שיטת שרשרת הדילוגים פותחה כדי לייעל את תהליך החיפוש בשרשרת. מבחינה לוגית השרשרת היא מבנה סדרתי פשוט ולכן אם יש בה N רשומות זמן החיפוש יהיה יחסי ל-N/2. בשיטה זו מחלקים את השרשרת לתת־קבוצות המכילות מספר רשומות קבוע ומשרשרים יחדיו את כל הרשומות הראשונות בכל קבוצה בסדר הפוך.



תרשים 8-16: מבנה שרשרת דילוגים

לצורך חיפוש מהיר נכנסים לשרשרת דרך כניסת הדילוג. בודקים את המפתח הדרוש לעומת המפתח הנמצא בכניסת הדילוג. אם המפתח הדרוש גדול יותר יש לפנות ולהתחיל בחיפוש סדרתי בתוך תת הקבוצה של השרשרת אשר יכיל Ng השוואות לכל היותר. אם המפתח הדרוש קטן מהמפתח בכניסת הדילוג, ניתן להשתמש במצביע הדילוג ולדלג לקבוצה הבאה ושוב לבצע את תהליך ההשוואה. הדילוג נעשה בכיוון של סדר יורד.

מספר ההשוואות במקרה זה יהיה $\sim\frac{1}{2}(Ng + \frac{N}{Ng})$

הגודל האופטימלי של תת־הקבוצות יהיה $Ng = \sqrt{N}$

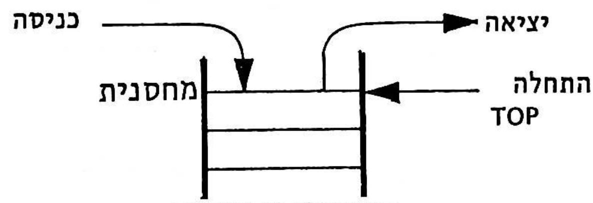
ההוכחה מופיעה בנספח לפרק זה.

שרשרת לינארית (Linear Chain)

שרשרת זו היא מבנה לינארי של נתונים הקשורים ביניהם בקבוצה סודרת אחת. הוספת רשומה חדשה או ביטול רשומה קיימת מתבצעת באחד הקצוות של השרישרת. המבנה הזה לא יכול להכיל או להצביע על שרשרות אחרות ואינו יכול להכיל לולאות.

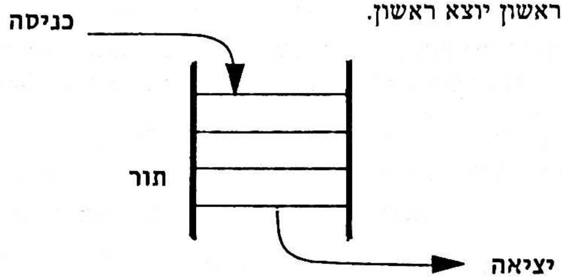
קיימים שלושה מבנים לינארים נפוצים:

א. מחסנית (Stack): שרשרת לינארית שבה כל ההוספות והביטולים מתבצעים בקצה אחד של השרשרת. מבנה זה מזכיר תור במבנה LIFO, כלומר האחרון שנכנס הוא הראשון שיוצא.



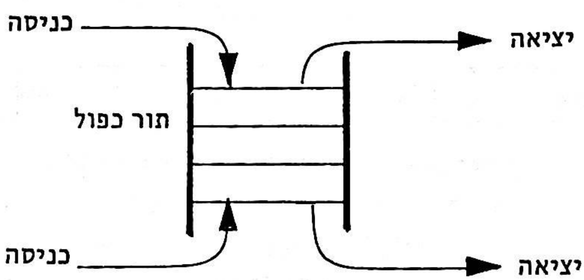
תרשים 8-17: מחסנית

ב. **תור** (Queue): שרשרת לינארית שבה ההוספות מתבצעות בקצה אחד והביטולים מתבצעים בקצה השני. מבנה זה מזכיר תור במבנה FIFO, כלומר מי שנכנס ראשון יוצא ראשון.



תרשים 8-18: מבנה תור

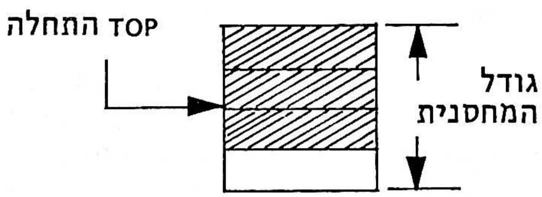
ג. **תור כפול** (Dequeue — Double ended queue): שרשרת שבה ההוספות והביטולים יכולים להתבצע בכל אחד מהקצוות.



תרשים 8-19: תור כפול

מבני הנתונים הלינארים חשובים מאד לניהול תורים של משימות בזכרון או לטיפול בשליפת נתונים כאשר מבנה הנתונים מורכב מאד. דוגמא לשימוש במחסנית נראה בפרק הדן במבנה עץ.

יישום השרשרות הלינאריות אפשרי גם על־ידי הקצאת מקום סדרתית או משורשרת. בהקצאת מקום סדרתית רושמים רשומה אחת לאחר השניה כמו בקובץ סדרתי.



תרשים 8-20: יישום מחסנית כמבנה לינארי

אם נניח שגודל הרשומה הוא R, המיקום (LOC) של רשומה חדשה יהיה:

$$LOC(X[i+1]) = LOC(X[i]) + R$$

שיטה זו נוחה מאד לטיפול במחסניות כי זקוקים רק למצביע לתחילת המחסנית (TOP). הוספת רשומה חדשה למחסנית Z תבוצע בצורה הבאה:

$$TOP \leftarrow TOP + R$$
$$Z[TOP] \leftarrow X$$

הוצאת רשומה מתוך המחסנית תבוצע בצורה הבאה:

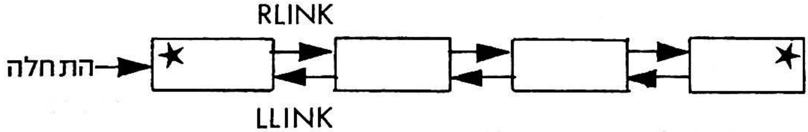
$$X \leftarrow Z[TOP]$$
$$TOP \leftarrow TOP - R$$

שיטת הקצאת מקום אחרת למחסנית היא על־ידי שימוש במבנה של שרשרת פשוטה. בשיטה הזאת אין קשר בין המקום הפיסי של הרשומה במחסנית לבין הסדר הלוגי שלה.



תרשים 8-21: יישום מחסנית בשרשרת פשוטה

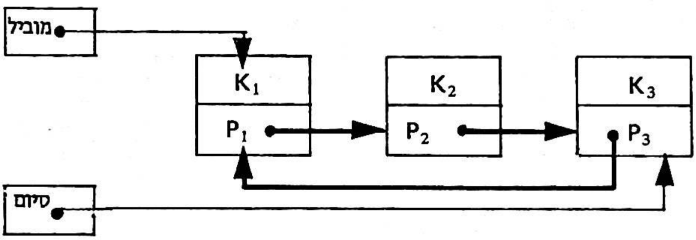
בשיטה זו מתקבלת גמישות רבה בניהול הקצאת המקום לניהול התור. אין חובה להשתמש דוקא בשרשרת חד־כיוונית. לניהול תור כפול דרושים שני מצביעים, אחד לתחילת התור ואחד לסופו.



תרשים 8-22: יישום מחסנית בשרשרת דו־כיוונית

8.5 טבעת פשוטה (Ring)

מבנה הטבעת דומה למבנה השרשרת בהבדל אחד. המצביע של הרשומה האחרונה בשרשרת מכיל את הכתובת של הרשומה הראשונה בשרשרת.



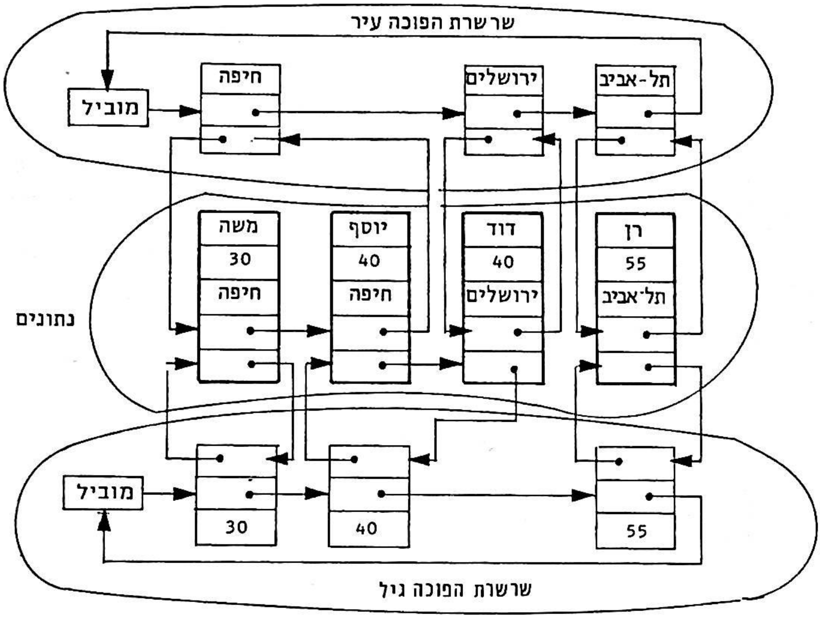
תרשים 8-23: מבנה טבעת

היתרון של מבנה טבעת הוא באפשרות הניתנת לסרוק את השרשרת בשני הכיוונים. אם מתחילים ברשומה האחרונה במבנה ניתן לבצע סריקה בכיוון לתחילת השרשרת. יתרון נוסף הוא האפשרות להתחיל לסרוק את המבנה בכל אחת מהרשומות. אם מגיעים בצורה כלשהי לרשומה K_2, על־ידי גישה ישירה למשל, ניתן להתחיל את הסריקה מרשומה זו ללא כל קושי. ניתן אם כן להגדיר את הטבעת כמבנה שרשרת ללא נקודת התחלה או נקודת סוף מיוחדת.

מבנה טבעת הינו אחד המבנים החשובים ביותר ונמצא בשימוש ברוב המערכות לניהול בסיסי נתונים. במערכות אלה רשומה אחת יכולה להשתתף במספר רב של טבעות שונות, דבר המאפשר ביצוע סריקות יעילות בקובץ במספר רב של חתכים.

בתרשים 8-24 מודגמת רשומת עובד המשתתפת בשתי טבעות הפוכות — טבעת "ערים" וטבעת "גיל". ניתן לסרוק את רשומות העובד הן לפי סדר אלפביתי של עיר המגורים או לפי סדר עולה של גיל העובד. כמו כן ניתן לענות לשאילתות בצורה יעילה על־ידי ביצוע חיתוך בין שתי הטבעות ללא גישה לרשומות.

בטבעת זמן החיפוש יחסי ל־N/2, כלומר למחצית מספר הרשומות. כאשר הטבעת היא כפולה ניתן לצמצם את זמן החיפוש. אם מגיעים לרשומה כלשהי בצורה ישירה והטבעת ממוינת, ניתן להחליט באיזה כיוון לבצע את החיפוש — קדימה או אחורה. במקרה זה זמן החיפוש הממוצע יהיה N/4.



תרשים 8-24: דוגמא לשיתוף רשומה בשתי טבעות

סריקה לפי מספר מפתחות (Multiple Key Search)

אם נשאלת השאלה: "מהו מספר העובדים אשר גרים בתל־אביב וגילם 55", עומדות בפנינו שתי אפשרויות:

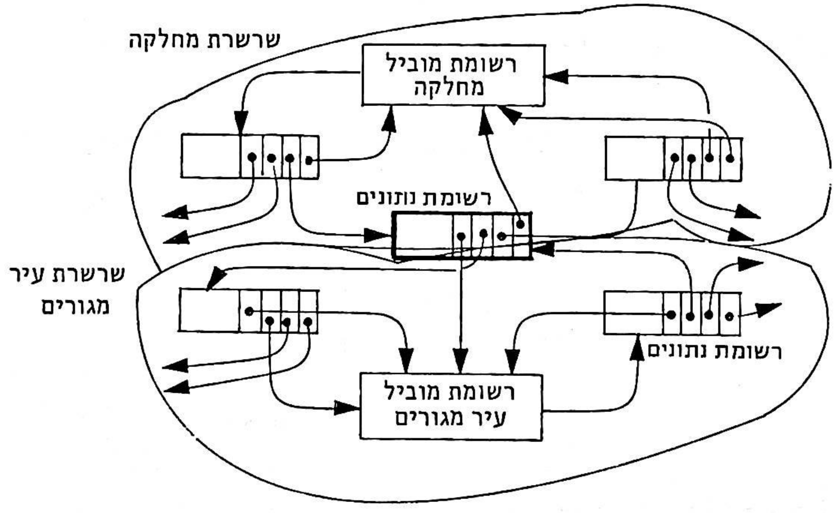
1. להתחיל את הסריקה לפי שרשרת "עיר מגורים", להגיע עד לרשומה של "תל־אביב" ומשם להתחיל את סריקת רשומות כל העובדים הגרים בתל־אביב. עבור כל עובד נבדוק את גילו ונשלוף רק את אלה שגילם 55.
2. להתחיל את הסריקה לפי שרשרת "גיל", להגיע עד לרשומה של גיל 55 ולהתחיל את סריקת רשומות העובדים הממלאים תנאי זה. עבור כל עובד נבדוק אם עיר מגוריו היא תל־אביב, שזהו התנאי השני בשאלה.

אילו היינו יודעים מהו מספר הרשומות המשתתפות בכל אחת מהשרשרות אפשר היה להחליט איזו משתי האפשרויות עדיפה. למטרה זו, מוסיפים בדרך כלל את "מונה מספר הרשומות בשרשרת" לרשומת המוביל של השרשרת. בחירת השרשרת עם מספר הרשומות הקטן ביותר תאפשר ביצוע מספר ההשוואות הקטן ביותר. כפי שהוצג בדוגמא, מונה זה מאפשר מתן תשובות מהירות על שאלות כגון:

— מהו מספר העובדים במפעל בגיל 55;
— מהו מספר העובדים במפעל הגרים בתל־אביב;
— ועוד.

8.6 מצביע לרשומת מוביל (Header Record Pointer)

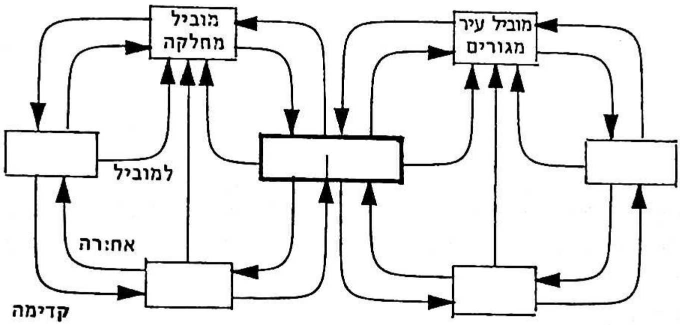
רשומת המוביל יכולה להכיל מידע נוסף לכתובת של הרשומה הראשונה בשרשרת. כדי לחסוך במקום אחסון ברשומות הקובץ ולאפשר עדכון נוח, ניתן להחזיק חלק מהנתונים הדרושים בקובץ ברשומת המוביל בלבד. במקרה זה יש לאפשר לכל רשומה בשרשרת להצביע אל רשומת המוביל כדי לקבל ממנה את הנתונים המשלימים. בדוגמא שבתרשים 8-25 נמצא מספר המחלקה ברשומת המוביל של שרשרת המחלקה בלבד. אם הרשומה משתתפת במספר שרשרות ניתן להגיע אליה דרך שרשרת שונה משרשרת המחלקה, ובמקרה זה אפשר לדעת את מספר המחלקה רק לאחר פנייה לרשומת המוביל.



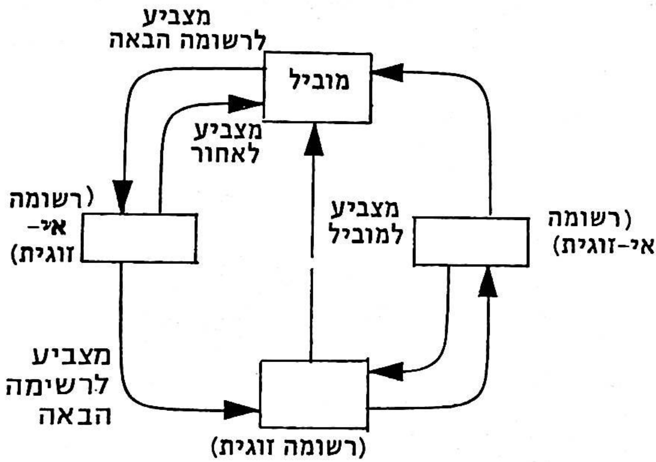
תרשים 8-25: שיתוף רשומה בשתי שרשרות עם הצבעות למוביל

ניתן להשתמש גם במבנה טבעתי דו־כיווני כדי לאפשר סריקה בשני הכיוונים ולהתחיל את הסריקה בכל רשומה בטבעת. בדוגמא שבתרשים 8-26 יש בכל רשומה שלשה מצביעים — קדימה, אחורה ולמוביל.

על מנת לחסוך במקום שדרוש עבור המצביעים קיימת אפשרות לשימוש בשני מצביעים בלבד בכל רשומה. מצביע אחד מורה תמיד על הרשומה הבאה בטבעת. המצביע השני מתחלף לסירוגין — ברשומה אחת הוא מורה על הרשומה הקודמת וברשומה הבאה על המצביע למוביל. מבנה זה קרוי **טבעת CORAL**.



תרשים 8-26: מבנה טבעתי עם הצבעות למוביל



תרשים 8-27: מבנה CORAL

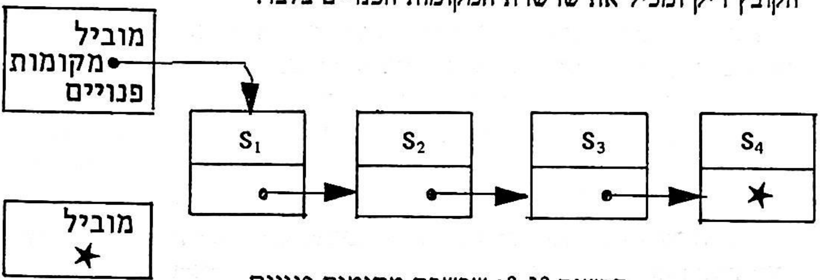
הרשומות האי־זוגיות מצביעות אחורה והרשומות הזוגיות מצביעות לרשומת מוביל. בתוך הרשומה יש לציין מה משמעות המצביע. לאיתור רשומת מוביל יש שני מצבים:

— אם נמצאים ברשומה זוגית יש מצביע ישיר,

— אם נמצאים ברשומה אי־זוגית יש לפנות אחורה לרשומה זוגית ומשם להשתמש במצביע.

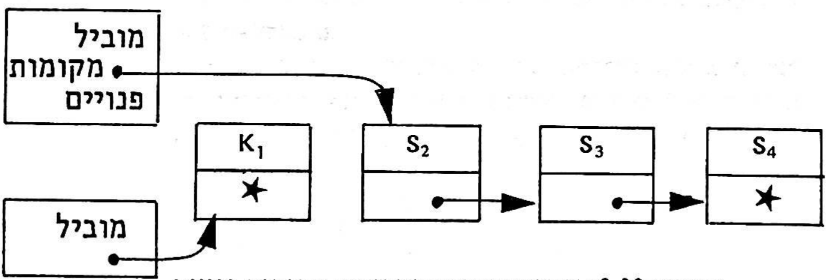
8.7 שרשרת מקומות פנויים (Free-Space List)

ברוב המערכות המיישמות שיטות ארגון משורשרות קיים מנגנון המטפל בניהול המקומות הפנויים בקובץ. מנגנון זה מאפשר קביעת מקום האחסון של רשומות חדשות במקומות פנויים ובמקום רשומות מבוטלות כדי שהקובץ לא ילך ויגדל עם הזמן. אחת האפשרויות לניהול המקומות הפנויים היא באמצעות שרשרת המקומות הפנויים. כאשר בונים את הקובץ משרשרים את כל הרשומות הריקות אשר יוצרות את השרשרת. בתרשים 8-28 מוצג מצב שבו הקובץ ריק ומכיל את שרשרת המקומות הפנויים בלבד.



תרשים 8-28: שרשרת מקומות פנויים

הוספת רשומה חדשה לקובץ גורמת לניתוק הרשומה הראשונה בשרשרת המקומות הפנויים והוספתה לשרשרת הנתונים.



תרשים 8-29: שרשרת נתונים ושרשרת מקומות פנויים

ביטול רשומת נתונים גורמת להחזרת הרשומה לשרשרת המקומות הפנויים. קיימות מספר אפשרויות להוספת המקום הפנוי לשרשרת: ניתן לצרף אותה לסוף השרשרת, או לתחילתה. כאשר הרשומות מצטרפות לתחילת שרשרת המקומות הריקים, בסוף הקובץ מנוצלים לעיתים רחוקות ורוב הרשומות מתרכזות בתחילת הקובץ. יש כאן שיפור מסוים מבחינת הזזות הזרוע של הדיסק.

8.8 תכנון שרשרות

מתכנן המערכת חייב להחליט באיזו מהשיטות לניהול שרשרות לבחור ובאיזו שרשרת לשתף כל רשומה. ההחלטה הזאת דומה להחלטה שבה קובעים איזה אינדקסים משניים לבנות. תהליך סריקת השרשרת דומה לסריקה בקובץ סדרתי ולכן מספר החיפושים יחסי ל־N/2. בפרק על אינדקסים ראינו שזמן החיפוש יחסי ללוגריתם של מספר הרשומות ולכן יש חשיבות רבה מאד לבחירת השרשרות והאינדקסים.

אחד השיקולים החשובים ביותר בזמן תכנון מבנה נתונים המבוסס על שימוש במצביעים, הוא אפשרות ההתאוששות במקרה של תקלה כלשהי במצביעים. התקלה יכולה לנבוע הן מבעית חומרה (תקלה ביחידת האיחסון למשל) או מבעית תוכנה (תוכנית ההורסת את תוכן המצביע) הצורך באפשרות התאוששות מהירה מתקלה הינו אחד הנימוקים העקריים לשימוש בשרשרות דו־כיווניות.

אם נשתמש כקריטריון שרשור בתכונה הקיימת בכל הרשומות תתקבלנה שרשרות ארוכות מאד. אם נשרשר למשל את כל רשומות העובדים הגברים במפעל בו רוב העובדים גברים, נמצא שרוב העובדים ישתתפו בשרשרת ולכן היא תהיה ארוכה מאד ולא יעילה בזמן חיפוש. אם נשתמש בתכונות המופיעות במספר קטן יחסית של רשומות נקבל באופן טבעי מספר שרשרות קצרות יחסית. לדוגמא, אם יש 20 מחלקות במפעל נקבל 20 שרשרות שונות שהן קצרות יותר משרשרת המפעל כולו. מספר החיפושים שנבצע במקרה זה דומה מאד לשימוש באינדקסים.

זמן חיפוש ממוצע: החיפוש בשרשרת מתבצע מתחילת השרשרת ועד למציאת הרשומה המבוקשת, תוך שימוש במצביעים. נניח שיש N רשומות בשרשרת. אם ההסתברות לחיפוש רשומה i היא P_i זמן החיפוש יהיה יחסי ל-

$$C = \sum_{i=1}^{N} i \cdot P_i$$

אם כל הרשומות מבוקשות באותה הסתברות, כלומר $P_i = 1/N$, נקבל זמן חיפוש ממוצע

$$\boxed{\overline{C} = \frac{N+1}{2}}$$

בדרך כלל נקבע הסדר של הרשומות בשרשרת בהתאם למפתח הלוגי שלהן ולכן השרשרת ממויינת. אם יש יותר ממפתח אחד לא ניתן למיין את השרשרת לפי שני מפתחות. אם ההסתברות P_i אינה שווה עבור כל i, ניתן לסדר את הרשומות בשרשרת בסדר כזה ש"מחיר" החיפוש יהיה מינימלי. לשם כך יש ל"קרב" לתחילת השרשרת את הרשומות המבוקשות יותר ובכך להקטין את מספר ההשוואות הדרושות. סידור הרשומות בשרשרת לפי סדר יורד של הסתברויות הביקוש שלהן יגרום למחיר חיפוש קטן יותר.

הבעיה היא שבמציאות הסתברויות אלה אינן ידועות. קימת אפשרות של ניהול מונה בכל רשומה המונה את מספר הגישות לרשומה ולבצע ארגון מחדש לקובץ לפי תוכן מונים אלה. שיטה זו מחיבת ביצוע ארגון מחדש של הקובץ על בסיס תקופתי ובזבוז מקום איחסון לניהול המונים. אפשרות אחרת היא לבצע ארגון מחדש של הקובץ על בסיס דינמי, כלומר תוך כדי פעולת הקובץ (Self-Organizing file). בשיטה זו, כל פעם שמתבצעת גישה לרשומה, הרשומה מועברת לתחילת השרשרת. המטרה היא שהרשומות שתדירות הגישה אליהן היא הגבוהה ביותר תמצאנה קרוב לתחילת השרשרת.

8.9 סיכום

יתרונות

— קיימת הפרדה בין המבנה הפיסי לבין המבנה הלוגי. הפרדה זו מפשטת מאד את עדכון הקובץ וחוסכת את כל הטיפול בגלישות.
— ניתן לשרשר את אותה רשומה במספר רב של שרשרות ובכך לאפשר את הגישה אליה במספר חתכים ללא צורך באינדקסים.
— קיים ניצול מחדש של מקומות שהתפנו.
— לא דרוש תרגום מפתח לכתובת ולכן ניתן לטפל בקלות יחסית ברשומות באורך משתנה.

חסרונות

— אם יש מספר רב של נקודות כניסה לקובץ יש לנהל אינדקס לשליטה על כניסות אלו. אינדקס זה אינו מתעדכן עם כל שינוי בשרשרת, למעט אותם מקרים בהם השינוי הוא בהתחלת או בסוף השרשרת.
— הרס של מצביע עלול לפגוע במבנה הלוגי של הקובץ. על מנת לאפשר שחזור יש להשתמש במצביעים נוספים.
— אורך רשומה יותר גדול מהנדרש עבור הנתונים בלבד מאחר ועליה להכיל גם את המצביעים.
— אין אפשרות להפעיל חיפוש בינרי על שרשרת. החיפוש בשרשרת דומה לחיפוש בקובץ סדרתי.
— יש לנהל שרשרת של מקומות פנויים.

8.10 שאלות

1. מדוע לא קיימת בעית גלישה במבני נתונים משורשרים?
2. הסבר את מושג המצביע. מה הוא מכיל?
3. מדוע משתמשים במצביעים קדימה ואחורה?
4. כתוב אלגוריתם לביטול רשומות במבנה טבעתי דו־כיווני.
5. כתוב אלגוריתם לחיפוש רשומה במבנה שרשרת עם תת־שרשרת. המפתח מורכב משני חלקים: זיהוי הרשומה בשרשרת וזיהוי הרשומה בתת־שרשרת.
6. הסבר את ההבדל בין שרשרת הפוכה לאינדקס משני.

8.11 נספח — חישוב גודל קבוצה אופטימלי בשרשרת דילוגים

שרשרת דילוגים מחולקת לקבוצות בעלות N_g רשומות כל אחת, לפיכך מספר הקבוצות בקובץ יהיה: $G = [\frac{N}{Ng}]$

הסימון [ ] מבטא את המספר השלם הגדול הקרוב ביותר לתוצאה, למשל 2 = [1.2]. במקרה של הסתברות שווה לחיפוש רשומה, מספר הדילוגים הממוצע יהיה:

$$\sum_{i=1}^{G} i \frac{1}{G} = \frac{G+1}{2}$$

לאחר שמגיעים לקבוצה מתאימה ממשיכים לסרוק את הרשומות בצורה סדרתית. מספר השוואות ממוצע בתוך הקבוצה יהיה:

$$\sum_{j=0}^{Ng-1} j \frac{1}{Ng} = \frac{Ng-1}{2}$$

מספר ההשוואות הכולל c בקובץ יהיה:

$$C = \frac{G+1}{2} + \frac{Ng-1}{2} = \frac{G+Ng}{2} = \frac{1}{2}(Ng + [\frac{N}{Ng}])$$

C תלוי בשני גורמים: ב־N מספר הרשומות בקובץ וב־N_g מספר הרשומות בקבוצה, כאשר האחד עולה השני יורד. נמצא מכאן את הגודל האופטימלי של הקבוצה:

$$\frac{dC}{dg} = \frac{1}{2}\left(1-\frac{N}{Ng^2}\right) = 0$$

$$Ng = \sqrt{N} \;;\; G = \sqrt{N} \;;\; C = \sqrt{N}$$

קיבלנו שאם נחלק את השרשרת לקבוצות בעלות $\sqrt{N}$ רשומות נקבל מחיר חיפוש אופטימלי אשר יהיה $\sqrt{N}$.

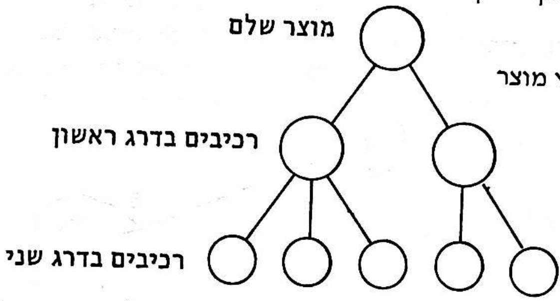
פרק תשיעי
ארגון נתונים מורכב — עצים
(Tree - Structured Data)

בפרק זה נציג מבנים מיוחדים של נתונים המשתמשים בשיטות ארגון לא לינאריות.

9.1 מבוא

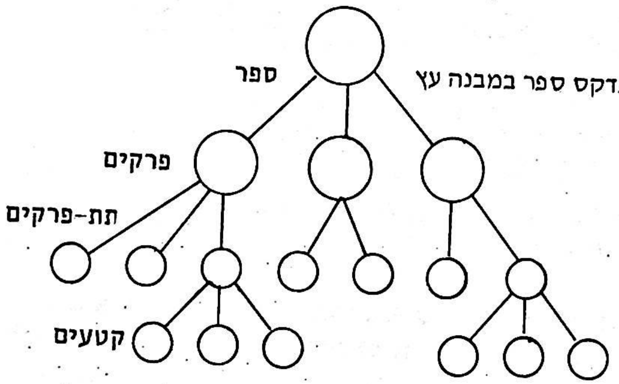
השיטות לארגון נתונים שסקרנו עד עתה מטפלות במבנה נתונים לינארי. קיימים יישומים בהם המבנה הלינארי של הנתונים אינו מתאים. כדי להמחיש זאת, נשתמש בדוגמאות.

כל מוצר מורכב מרכיבים שהם בעצמם יכולים להיות מורכבים מרכיבים אחרים וכך הלאה. תיאור גרפי של רכיב המתפצל לרכיבים אחדים בשרשרת יוצר מבנה עץ הפוך, הקרוי עץ־מוצר או Bill-Of-Material.



תרשים 9-1: מבנה עץ מוצר

תוכן ענינים של ספר מכיל הפנייה לפרקים, לתת־פרקים ולקטעים בתוך תת־פרק. תיאור גרפי של תוכן הענינים מציג גם־כן מבנה עץ.



תרשים 9-2: אינדקס ספר במבנה עץ

היתרון הגדול של מבנה העץ הוא בניצול היעיל של שטח האחסון. למעשה יש כאן אינדקס ונתונים המעורבים יחדיו. התוכן של המפתח קובע את מיקום הצומת בעץ. בתוך הצומת עצמו נמצא את הנתונים הדרושים ללא צורך לגשת לקובץ נתונים נפרד. תהליך החיפוש מהיר מאד אם נסדר את הרשומות בצמתים לפי סדר מסוים. מבנה העץ משלב את מהירות החיפוש הבינרי עם פשטות העדכון של מבנה משורשר.

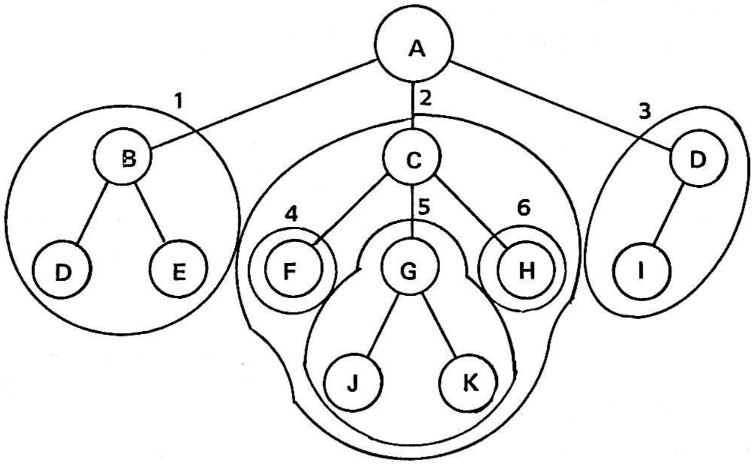
9.2 ארגון נתונים במבנה עץ

נציג תחילה את הגדרת מבנה עץ.

הגדרת עץ — העץ מורכב **מצמתים** (Nodes) **וחיצים** (Arrows).

1. הצומת הראשי בעץ קרוי **שורש** (Root).
2. אוסף של **חיצים** מכוונים מובילים מהשורש לצמתים אחרים.
3. כל צומת הוא שורש של **תת־עץ** של העץ הראשי. תת־עץ יכול להכיל שורש בלבד, או שורש וצמתים נוספים.

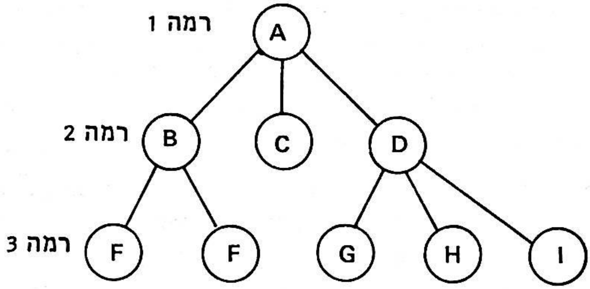
כפי שניתן לראות הגדרת העץ היא רקורסיבית, כלומר הגדרנו את העץ במונחים של עץ.



תרשים 9-3: חלוקת עץ לתת־עצים

בדוגמא שבתרשים 9-3 יש לשורש A שלושה תת־עצים: 1, 2, 3. לתת־עץ 2 למשל, יש שורש C. לשורש זה יש גם כן שלושה תת־עצים: 4, 5, 6.

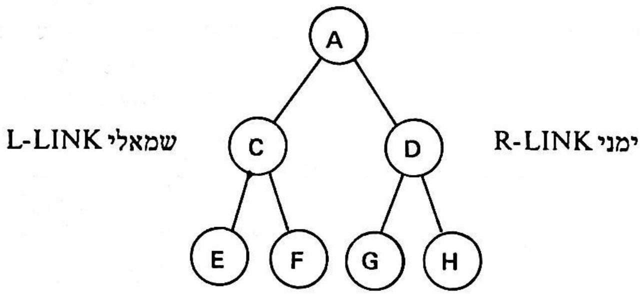
למספר תת־העצים השייכים לצומת מסוימת קוראים **דרגת הצומת** (Node Degree). לעץ יש מספר רמות כאשר רמת השורש תסומן ב־1. רמה 2 תכיל את תתי־העצים הקשורים לשורש. רמה 3 תכיל את תתי־העצים הקשורים לרמה 2, וכן הלאה. ראה תרשים 9-4.



תרשים 9-4: רמות הצמתים בעץ

עצים בינריים

עץ בינרי מוגדר כעץ המורכב מצמתים אשר הדרגה שלהם לא גבוהה מ־2, כלומר כל צומת יכול להצביע על שני תת־עצים לכל היותר, הקרויים במקרה זה תת־עץ ימני (R-LINK) ותת־עץ שמאלי (L-LINK).



תרשים 9-5: מבנה עץ בינרי

סריקת רשומות במבנה עץ בינרי

אחת הבעיות החשובות בניהול הקובץ היא סריקת העץ. דרושה שיטה לסריקת כל העץ שבה נעבור בכל צומת פעם אחת בלבד. קיימות שלש שיטות לסריקת עצים. נציג את תהליך הסריקה בכל אחד מהן על פי הדוגמא שבתרשים 9-5.

1. Pre-Order
   א. שורש
   ב. תת־עץ שמאלי
   ג. תת־עץ ימני

עבור הדוגמא נקבל את סדר הסריקה הבא: A, C, E, F, D, G, H

2. In-Order
   א. תת־עץ שמאלי
   ב. שורש
   ג. תת־עץ ימני.

עבור הדוגמא נקבל את הסדר הבא: E, C, F, A, G, D, H

3. Post-Order א. תת־עץ שמאלי
ב. תת־עץ ימני
ג. שורש

עבור הדוגמא נקבל את הסדר הבא: E, F, C, G, H, D, A

תאור אלגוריתם לסריקת Pre-Order

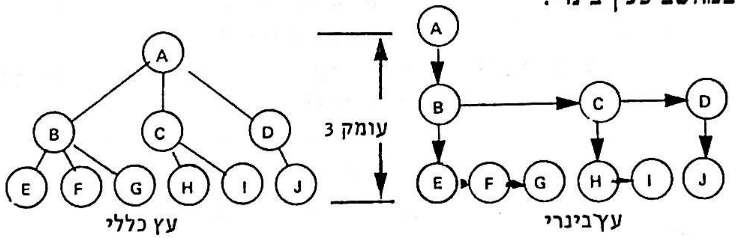
מתחילים את הסריקה בשורש A. לאחר מכן יש לבקר בתת־עץ השמאלי שלו. מאחר ואין מצביע חזרה מהתת־עץ לשורש יש לשמור את הכתובת של A כדי שאפשר יהיה לעבור אחר כך לתת־עץ הימני שלו. ניתן להשתמש במחסנית ונרשום בה את הכתובת של A. נתקדם עכשיו לצומת C. אחר כך יש לעבור לצומת E, אבל תחילה יש לשמור את הכתובת של C ונרשום גם אותה במחסנית. לאחר סריקת E נקרא את האיבר הראשון במחסנית (שהוא הכתובת האחרונה שנכתבה — C) ונעבור לתת־עץ הימני של C. המחסנית מיועדת לשמירת כתובות אשר צריך לפנות אליהן שוב בתהליך הסריקה. המחסנית מופעלת בשיטת LIFO.

ננסח עתה את האלגוריתם בצורה פורמלית, ונסמן ב־P את הכתובת אליה אנו פונים וב־T את התחלת העץ, כלומר את כתובת השורש. התהליך של הסריקה מוצג בצורה הבאה:

1. התחל — אפס מחסנית. קבע P = T.
2. אם P = ⋆ עבור ל־4.
2. הכנס כתובת הנמצאת ב־P לתוך מחסנית S. קבע P = LLINK וחזור לצעד 2.
4. אם המחסנית S ריקה סריקת העץ הסתיימה. אחרת, יש להעביר את האיבר הראשון מתוך המחסנית אל P.
5. ביקור בצומת — גש לרשומה אשר כתובתה ב־P ובצע את העיבוד הדרוש. קבע P = RLINK וחזור לצעד 2.

ייצוג עצים כלליים

ייצוג עצים בינרים במחשב נוח מאד מאחר וכל רשומה מכילה בדיוק שני מצביעים, אחד לתת־עץ השמאלי ואחד לתת־עץ הימני. אם נרצה לאחסן במחשב עצים כלליים נראה שמספר המצביעים בכל צומת משתנה וזה מוסיף סיבוך רב למבנה ולתהליך החיפוש. למען הפשטות ניתן לייצג עץ כללי במחשב כעץ בינרי.



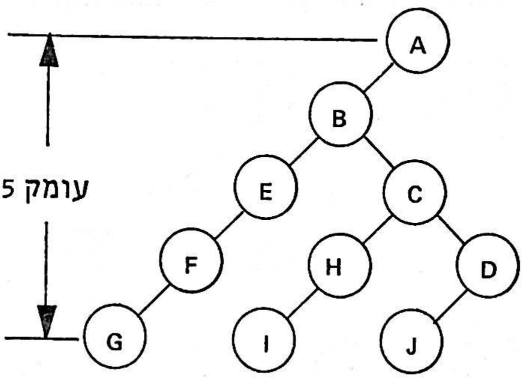
תרשים 9-6: הפיכת עץ כללי לעץ בינרי

על מנת להפוך עץ כללי לעץ בינרי כל צומת תצביע על:

— צומת הבן הראשון, כלומר הצומת השמאלי ברמה הבאה.

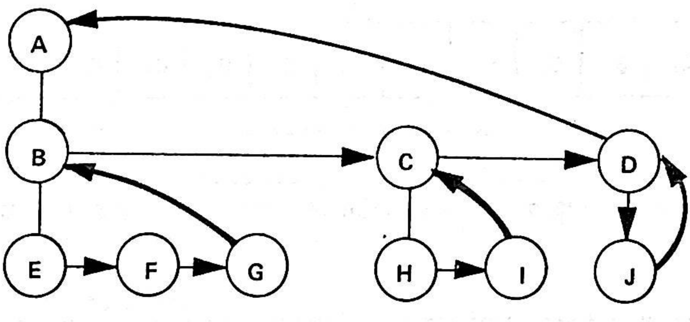
— צומת התאום שלו, כלומר הצומת השכן באותה רמה.

במעבר הזה מאבדים מספר קשרים ישירים, כמו בין צומת B ל-G למשל. עכשיו נצטרך לעבור דרך E ו-F כדי להגיע ל-G. העומק של העץ (מספר הרמות המקסימלי שלו) גדל. בתרשים 9-6 עברנו מעץ בעומק 3 לעץ בעומק 5. תרשים 9-7 מציג את העץ הכללי בצורת עץ בינרי.



תרשים 9-7: עץ בינרי המתקבל מהעץ הכללי

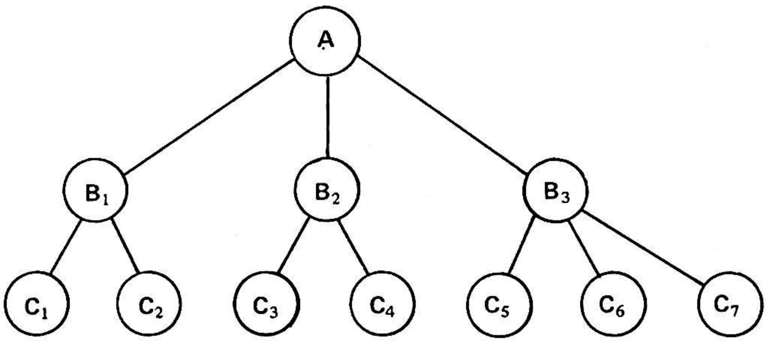
מאחר ובצמתים סופיים אין מצביע לבנים אחרים ניתן לנצל מקום זה למצביע המורה חזרה אל צומת האב כפי שמוצג בתרשים 9-8. מבחינה לוגית עדיין יש עץ בינרי אך את המצביעים הפנויים מנצלים ליצירת מבנה טבעתי אשר יכול ליעל את תהליכי החיפוש במבנה. יש צורך לעשות אבחנה בין מצביע המורה לבן לבין מצביע המורה חזרה לצומת האב.



תרשים 9-8: הוספת מצביעים לקבלת מבנה טבעתי

9.3 שיטות לייצוג עצים ביחידות אחסון

הדיון הקודם במבנה העצים צריך להנחות לשיטה של ייצוג רשומות הקובץ ביחידות אחסון. נשתמש בדוגמת עץ המוצגת בתרשים 9-9 שבו כל אחת מהאותיות בעיגול (צומת) מסמלת רשומה. נסקור עתה את השיטות המקובלות לאחסנת קובץ העץ.



תרשים 9-9: דוגמא לעץ כללי

א. אחסון רציף

הרשומות מאוחסנות כרצף של צמתים מלמעלה למטה ומשמאל לימין. שיטה זו מתאימה לעיבודי אצווה מאחר וכל הבנים ברמה הנמוכה ביותר מופיעים אחרי האב שלהם. לדוגמא, כל הבנים של B_2 נרשמים אחריו. שיטה זו מתאימה לשמירת עצים בסרט מגנטי למשל. בתוך הרשומה יש להוסיף שדה המציין את הדרמה של הצומת.

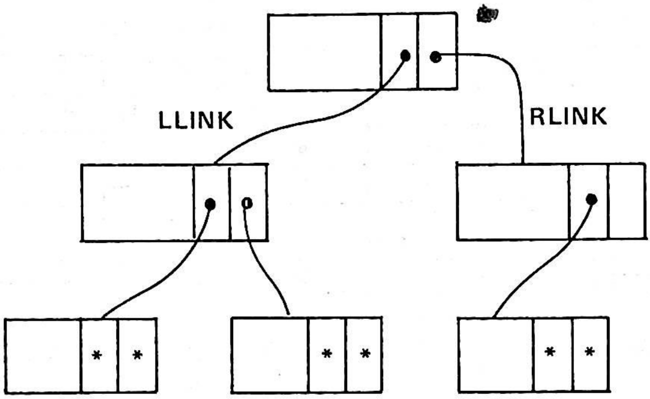
A_1	B_1	C_1	C_2	B_2	C_3	C_4	B_3	C_5	C_6	C_7

תרשים 9-10: ייצוג רציף

בשיטה זו קשה מאד לבצע עדכונים מאחר והמקום הפיסי קובע את הסדר הלוגי של הרשומה. כמו כן אין אפשרות להגיע לרשומה מבלי לעבור תחילה דרך כל המסלול המוביל אליה.

ב. שימוש במצביעים

לכל רשומה מוסיפים שני מצביעים — אחד לבן הימני ואחד לבן השמאלי, ואת העץ הכללי הופכים לעץ בינרי עם זרועות שמאל וימין.

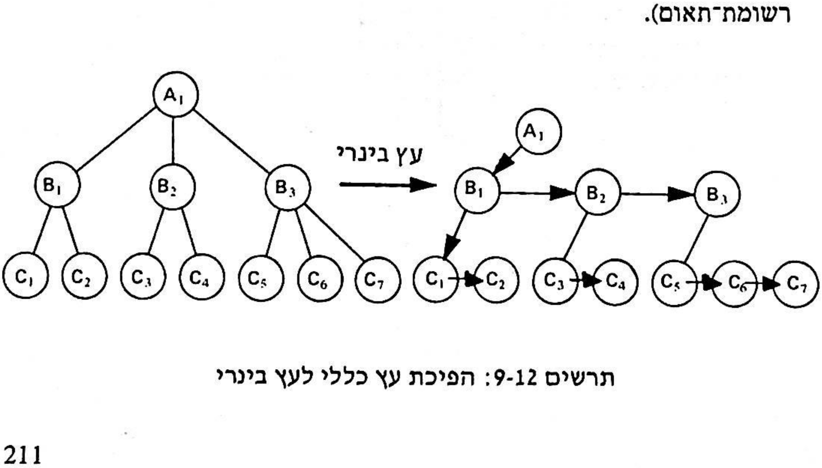


תרשים 9-11: שימוש במצביעים לאחסון עץ בינרי

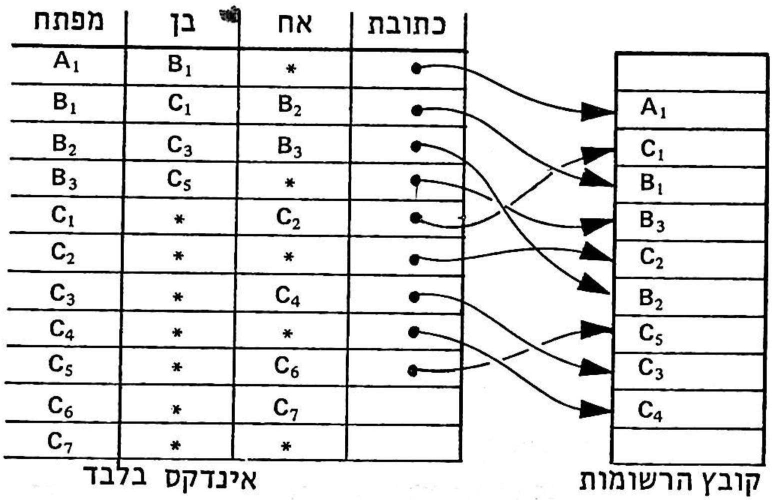
השימוש במצביעים מאפשר להפריד בין המיקום הפיסי של הרשומה לבין מקומה הלוגי בתוך העץ. יש לזכור שדרושה תוספת מקום בתוך הרשומה לשמירת המצביעים. בשיטה המוצגת, הנתונים והמצביעים מופיעים יחד באותו הקובץ אך קיימת אפשרות להוציא את המצביעים אל מחוץ לרשומות כפי שמוצג בהמשך.

ג. אינדקס המכיל את מבנה העץ

בונים אינדקס מיוחד, נפרד מהקובץ, אשר מכיל את מבנה העץ. הפרדה זו מאפשרת שהקובץ עצמו יכול להיות מאורגן בשיטה כלשהי. בניית האינדקס מכוונת לאפשר חיפוש מהיר של רשומות. תהליך העבודה כולל שני שלבים: הפיכת העץ הכללי לעץ בינרי ובניית האינדקס המכיל את מבנה העץ בלבד, כפי שמוצג בתרשימים 9-12 ו־9-13. נשתמש במושג חדש, **אח** עבור רשומה ברמה שוה, למשל B_2 היא רשומת־אח ל־B_1. (לעיתים משתמשים גם במושג רשומת־תאום).

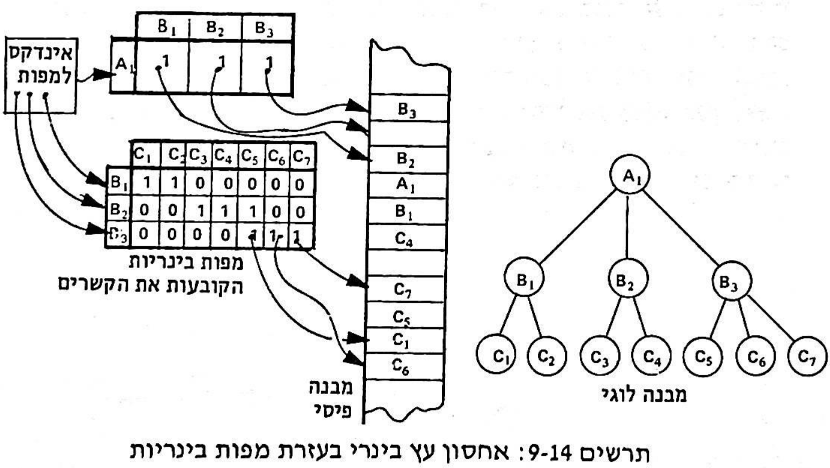


תרשים 9-12: הפיכת עץ כללי לעץ בינרי



תרשים 9-13: אחסון מבנה העץ באינדקס בלבד

בדוגמא זו אנו מדלגים בין הרשומות במבנה הלוגי של העץ תוך שימוש באינדקס. לחיפוש של הבנים של A_1 יש למצוא תחילה את הבן הראשון B_1 וממנו לעבור לאח שלו B_2 וממנו לעבור לאח B_3 עד שאין אחים נוספים.



תרשים 9-14: אחסון עץ בינרי בעזרת מפות בינריות

ד. מפות בינריות

ניתן לשמור מפה בינרית עבור הקשר בין כל אחת מהרמות של העץ בשיטה של מטריצות זיקה בין הרמות השונות. לכל רמה יש מטריצה נפרדת. ככל שיורדים ברמת העץ המטריצות גדולות יותר. המפות הבינריות יעילות מאד מבחינת מהירות החיפוש.

חישוב עומק העץ

עומק העץ מוגדר כמספר המקסימלי של קשתות שיש לעבור כדי להגיע לצומת היעד ברמה הגבוהה ביותר. לעומק העץ חשיבות רבה מאחר והוא מציין את מספר ההשוואות שצריך לבצע כדי להגיע לצומת היעד.

ניתן להוכיח שעומק העץ נתון לפי הנוסחה הבאה:

$$\boxed{H = \log_m N}$$

כאשר m מספר הבנים המקסימלי בצומת, ו־N הוא מספר הרשומות בעץ. ההוכחה לנוסחה מופיעה בנספח בסוף הפרק.

9.4 עצי חיפוש בינריים

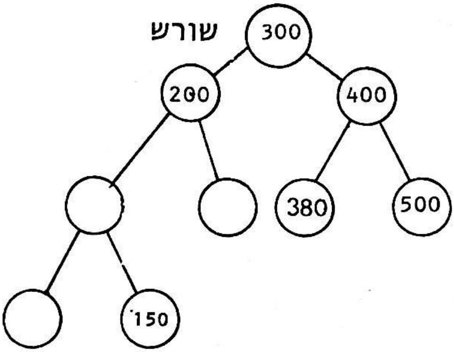
במבני העצים שהוצגו עד עתה לא היה כל קשר בין הערך של הצומת לבין המיקום היחסי שלו בעץ. בתהליכי חיפוש מעונינים לאתר את הצומת מידית ולא על־ידי סריקה לינארית של כל העץ. ניתן לבנות את העץ לפי סדר כלשהו כדי שהאיתור יהיה מהיר.

עץ חיפוש בינרי הוא עץ בינרי המקיים את התנאי הבא:

— כל צומת שמאלי לצומת בו נמצאים יהיה בעל ערך נמוך יותר.

— כל צומת ימני לצומת בו נמצאים יהיה בעל ערך גבוה יותר.

בתרשים 9-15 מוצג בתוך הצומת גם ערך הצומת לפי התנאי של עץ חיפוש בינרי. בכל צומת מתקיים התנאי: הבן הימני בעל ערך גבוה יותר מערך הצומת והבן השמאלי בעל ערך נמוך יותר מערך הצומת.



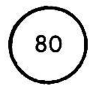
תרשים 9-15: עץ חיפוש בינרי

שיטה זו לבניית העץ תאפשר ביצוע חיפושים מהירים. לדוגמא נניח שמחפשים צומת בעלת הערך 150. הכניסה לעץ תהיה תמיד דרך השורש. נשווה את ערך השורש לערך הדרוש, ומאחר וערך השורש גבוה יותר, נפנה שמאלה ונגיע לצומת בעל הערך 200. נבצע שוב השוואה ונפנה פעם נוספת שמאלה ונגיע לצומת בעל הערך 100. נבצע השוואה ונפנה הפעם ימינה מכיון שצומת בערך גבוה יותר נמצא מימין. השוואה נוספת תאתר את הצומת המבוקש 150. מספר ההשוואות שבצענו הוא 4 וזה דומה לתהליך החיפוש הבינרי שהודגם בפרק הדן בארגון קבצים סדרתי: מספר ההשוואות שנבצע יהיה יחסי ל־$\log_2 N$.

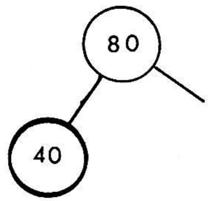
מבנה העץ הבינרי תלוי בצומת שנבחרת להיות צומת השורש ובסדר הגעת הצמתים (הרשומות) לעץ. על פי נוהל החיפוש נדגים שלבי בניית עץ חיפוש

בינרי בהנחה שמפתח הסדר הוא מספרי בלבד.

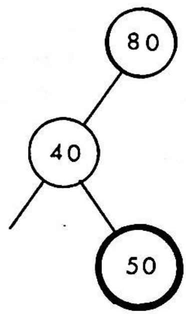
1. רשומה ראשונה עם מפתח 80.



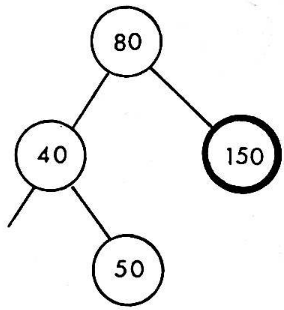
2. הוספת רשומה עם מפתח 40. מאחר והיא קטנה מ־80, היא תרשם מצד שמאל.



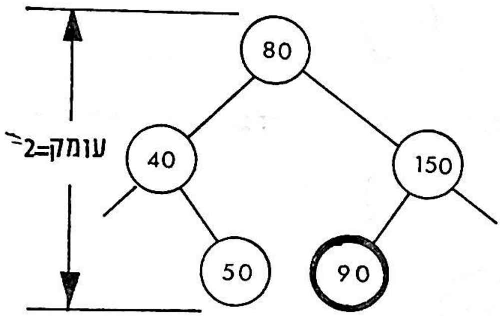
3. הוספת רשומה עם מפתח 50, מצד ימין של צומת 40.



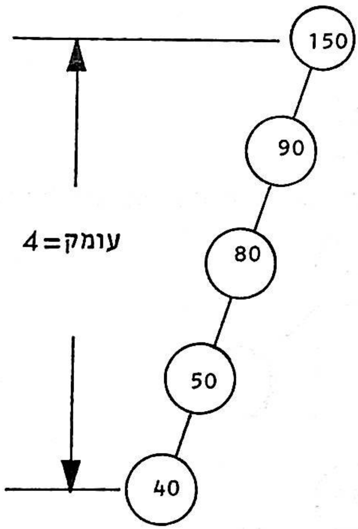
4. הוספת רשומה עם מפתח 150, בצד ימין של צומת 80.



5. הוספת רשומה עם מפתח 90, מצד שמאל של צומת 150.



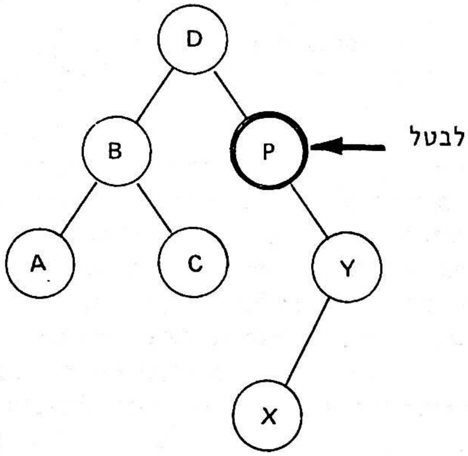
בצורה זו נמשיך ונוסיף רשומות חדשות לעץ. אם הרשומות היו מגיעות בסדר אחר, למשל: 150 ואח"כ 90, 80, 50, 40, כלומר רשומות ממוינות מסדר יורד, היינו מקבלים עץ השונה מהעץ הקודם. בעוד שהעץ הקודם היה בעומק 2 ולכל היותר נדרשו 3 השוואות וגישות כדי לשלוף רשומה כלשהי, הרי העץ החדש הוא בעומק 4, ובו נדרשות עד 5 גישות והשוואות לאיתור רשומה מבוקשת.



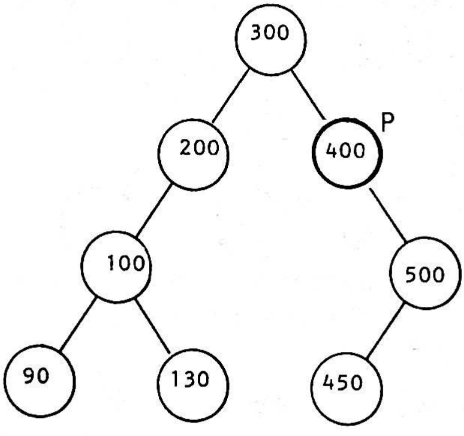
תרשים 9-16: בניית עץ חיפוש בינרי עם רשומות בסדר יורד

ניתן לראות שמבנה העץ תלוי ברשומה הראשונה שמגיעה, מאחר והיא הופכת להיות רשומת השורש. המבנה תלוי גם בסדר הגעת הרשומות ובסדר המיון שלהן.

עדכון עץ בין הינו תהליך מורכב מאחר וחייבים לשמור על סדר העץ ולא לשבש את תכונות החיפוש. להדגמה נשתמש בתרשים 9-17 שבו מבטלים את צומת P. לשם כך יש להפוך את עץ החיפוש הבינרי הזה לעץ חיפוש בינרי אחר הכולל את כל הצמתים למעט הצומת המבוטל. אפשר לעשות זאת באחת משתי דרכים: לבצע הזזה של החלק הימני של העץ או הזזה של החלק השמאלי של העץ.



תרשים 9-17: עדכון עץ בינרי



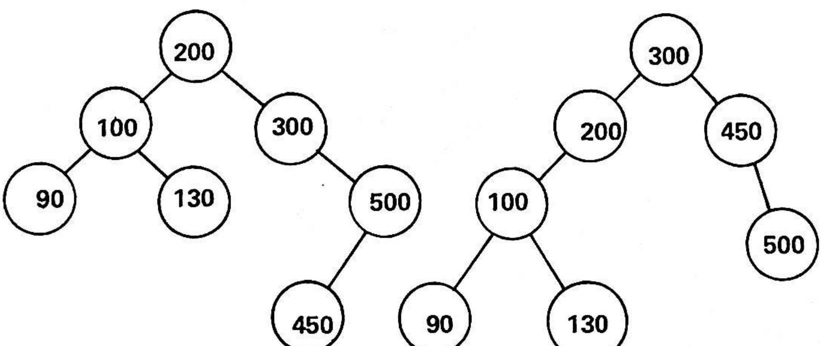
תרשים 9-18: ביטול צומת בעץ חיפוש בינרי

להצגת תהליך העדכון נשתמש בתרשים 9-18 שבו מתואר עץ עם ערכי צמתים. דרוש לבטל את צומת P שערכו המספרי 400. כפי שהוסבר יש להפוך את עץ החיפוש הזה לעץ חיפוש אחר הכולל את כל הצמתים למעט הצומת המבוטל P = 400.

כדי להפוך את מבנה העץ רושמים את כל המפתחות שלו לפי סדר In-Order, ומקבלים עץ ממויין. ובדוגמא שלנו:

90 ,100 ,130 ,200 ,300 ,400 ,450 ,500

להוצאת הצומת P מתוך העץ יש להזיז את כל הצמתים משמאל לו מקום אחד ימינה, או להזיז את כל הצמתים מימין לו מקום אחד שמאלה. בכל אחד מהמקרים מתקבל עץ חיפוש בינרי שונה, כפי שמוצג בתרשים 9-19. הזזת העץ ימינה גרמה לשינוי במספר רב של צמתים וגרמה גם שינוי מיקום ורמה. הזזת העץ שמאלה פשוטה יותר מאחר וזהו שינוי מקומי, בענף אחד של העץ. השינוי מתבטא בכך שבמקום הצומת המבוטל מציבים את הבן השמאלי אם הוא קיים, או את הבן השמאלי הראשון של אחד הבנים הימניים שלו. בדוגמא שבתרשים הצומת עם ערך 450 הינו הבן השמאלי הראשון של אחד הבנים הימניים של הצומת המבוטל. ביצוע הביטול בצורה זו הינו פשוט יחסית ובעל אופי מקומי בעיקרו.

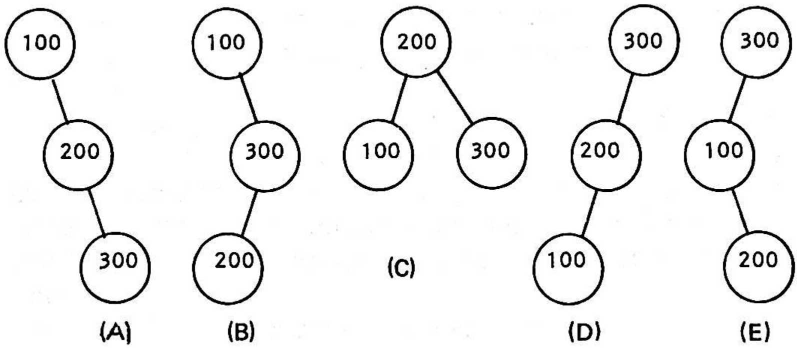


תרשים 9-19: שני עצים שונים המתקבלים מביטול צומת P

9.5 עצי חיפוש אופטימליים

בעצי חיפוש בינריים קיים סדר מסוים כלשהו של המפתחות בתוך העץ ומאוסף אחד של רשומות ניתן לבנות מספר רב מאד של עצי חיפוש. בניית עץ החיפוש מכוונת להשיג יעילות חיפוש בקובץ, ובנושא זה נדון בהמשך.

לדוגמא נתונות שלש רשומות עם המפתחות 100, 200, 300. מרשומות אלו ניתן לבנות מספר עצים שונים בהתאם לבחירת הרשומה בצומת השורש ולפי סדר ההגעה של שאר הרשומות. כמה מהאפשרויות מוצגות בתרשים 9-20.



תרשים 9-20: אפשרויות בנית עצי חיפוש בינריים

מי מתוך העצים שבתרשים 9-20 הינו עץ החיפוש האופטימלי? כדי לענות לשאלה זו יש לדעת תחילה את הסתברויות החיפוש של כל אחת מהרשומות. נניח שההסתברויות הן:

מפתח הרשומה	K_i	100	200	300
הסתברות	P_i	0.3	0.3	0.4

כלומר, לאורך זמן ההסתברות לחיפוש רשומה עם מפתח 200 היא 0.3.

כדי לבחור את העץ האופטימלי מבין חמשת עצי החיפוש האפשריים, יש לחשב את תוחלת מספר ההשוואות בכל אחד מהם ולבחור בעץ שעבורו תוחלת זו היא מינימלית.

מספר העצים האפשריים:

$$\min \left\{ C_{Nj} = \sum_{i=1}^{n} P_{ij} h_{ij} \right\}$$

$j = 1, 2, \ldots K$

h_i — עומק הצומת i בעץ j

C_{N_j} — מחיר החיפוש בעץ j כאשר יש N צמתים.

נבצע את החישוב עבור הדוגמא:

$$C_{NA} = 1 \cdot 0.4 + 2 \cdot 0.3 + 3 \cdot 0.3 = 1.9$$

$$C_{NB} = 1 \cdot 0.4 + 2 \cdot 0.4 + 3 \cdot 0.3 = 2.1$$

$$C_{NC} = 1 \cdot 0.3 + 2 \cdot 0.3 + 2 \cdot 0.4 = \underline{1.7}$$

$$C_{ND} = 1 \cdot 0.4 + 2 \cdot 0.3 + 3 \cdot 0.3 = \overline{1.9}$$

$$C_{NE} = 1 \cdot 0.4 + 2 \cdot 0.3 + 3 \cdot 0.3 = 1.9$$

לפי הסתברויות החיפוש שניתנו, תוחלת מספר החיפושים המינימלית התקבלה עבור העץ הסימטרי C שבו נבצע 1.7 השוואות בממוצע. אם ההסתברויות היו שונות, למשל 0.7, 0.2, 0.1 היינו מקבלים שהעץ האופטימלי הוא E.

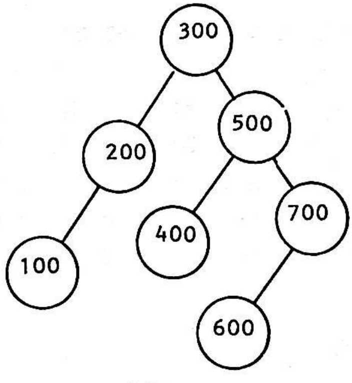
אם ההסתברות לחיפוש רשומות זהה, כלומר לפנינו התפלגות אחידה $Pi = 1/N$ העצים הסימטריים יהיו העצים האופטימליים.

עבור מספר קטן של צמתים ניתן לבנות את כל העצים האפשריים ולבחור מתוכם את העץ הטוב ביותר. ככל שמספר הצמתים גדל מספר העצים האפשריים גדל בכמה סדרי גודל ולכן דרושים אלגוריתמים לאיתור העץ האופטימלי.

קיימים מספר אלגוריתמים לבניית עצי חיפוש אופטימליים לפי Knuth‏ Hu and Tucker, ואחרים. עם זאת ניתן לנסות ולהשתמש באלגוריתמים פשוטים שאינם בונים עצים אופטימליים אך יכולים לבנות עצים שקרובים לאופטימלים. נציג כאן אלגוריתם מסוג זה המניח שרצוי לקרב לשורש צמתים בעלי הסתברות גבוהה לחיפוש. האלגוריתם הפשוט הזה יכלול את הפעולות הבאות:

1. מיון כל הצמתים לפי סדר עולה של ההסתברויות.
2. בניית עץ חיפוש בינרי לפי סדר הצמתים.

בתרשים 9-21 מוצגת בניית עץ החיפוש על פי האלגוריתם הפשוט שהודגם.



תרשים 9-21

Ki	300	500	200	400	700	100	600
Pi	0.2	0.2	0.15	0.15	0.1	0.1	0.1

תוחלת מספר ההשוואות בעץ זה תהיה:

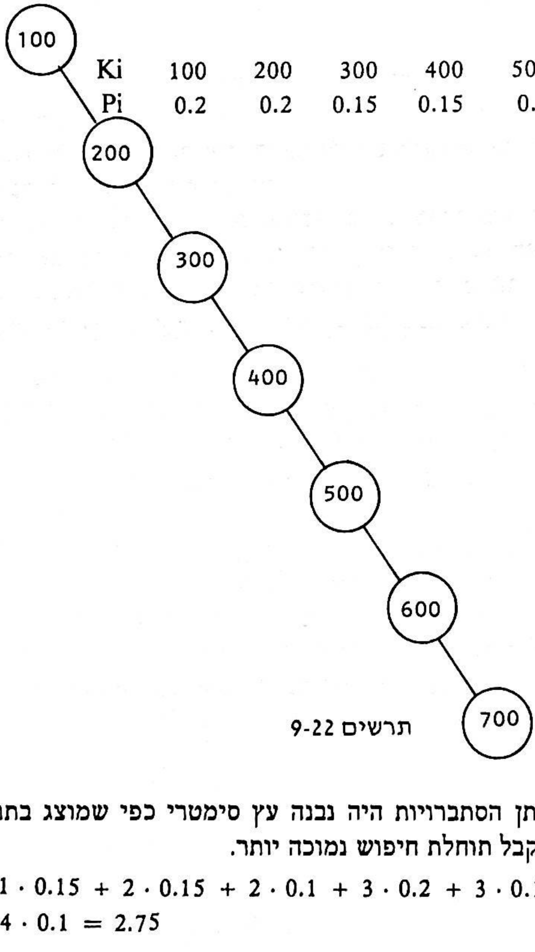
$$C_N = 1 \cdot 0.2 + 2 \cdot 0.2 + 2 \cdot 0.15 + 3 \cdot 0.15 + 3 \cdot 0.1 + \\ + 4.01 = 2.35$$

אלגוריתם זה פשוט ואינו עונה למקרה שבו לאחר המיון לפי הסתברויות גם ערכי הצמתים ממוינים לפי סדר עולה או יורד של מפתחות. במקרה זה נקבל עץ מנוון, כלומר קו משופע בכיוון ימין או שמאל שהוא בוודאי אינו עץ אופטימלי.

מצב שבו ערכי הצמתים ממויינים לאחר מיון ההסתברויות מוצג בתרשים 9-22. התוצאה היא עץ מנוון שבו תוחלת מספר ההשוואות הוא:

$$C_N = 1 \cdot 0.2 + 2 \cdot 0.2 + 3 \cdot 0.15 + 4 \cdot 0.15 + 5 \cdot 0.1 + 6 \cdot 0.1 + \\ + 6 . 0.1 + 7 . 0.1 = 3.45$$

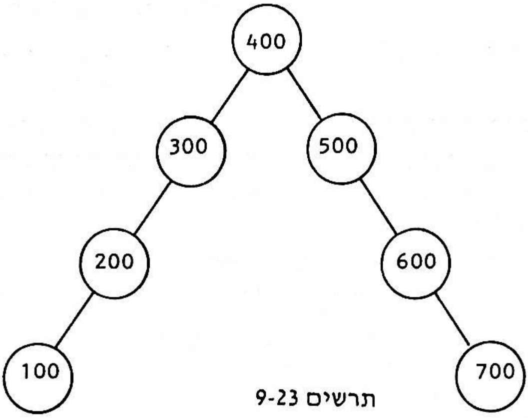
Ki	100	200	300	400	500	600	700
Pi	0.2	0.2	0.15	0.15	0.1	0.1	0.1



תרשים 9-22

אם על סמך אותן הסתברויות היה נבנה עץ סימטרי כפי שמוצג בתרשים 9-23 אפשר היה לקבל תוחלת חיפוש נמוכה יותר.

$$C_N = 1 \cdot 0.15 + 2 \cdot 0.15 + 2 \cdot 0.1 + 3 \cdot 0.2 + 3 \cdot 0.1 + 4 \cdot 0.2 + \\ 4 \cdot 0.1 = 2.75$$



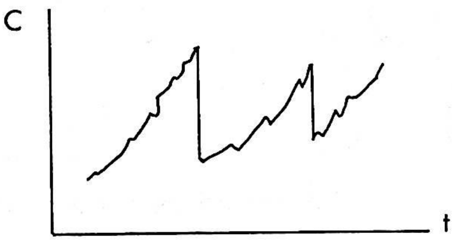
תרשים 9-23

על מנת להתגבר על בעיית ניוון העץ ניתן להשתמש באלגוריתם שונה במקצת. משתדלים להציב מימין ומשמאל של כל צומת צמתים חדשים שסכום ההסתברויות שלהם שווה ועל ידי כך יוצרים מרכז הכובד. אך גם באלגוריתם זה לא נקבל תמיד עצים סימטריים.

כפי שראינו עד כאן עצים סימטריים הם בדרך כלל עצי חיפוש טובים במיוחד אם ההסתברויות לכל הרשומות זהות. נעשו מאמצים לבנות עצי חיפוש סי.טריים ולהתגבר על הבעיה הגדולה ביותר — כיצד לשמור שימשיכו להשאר סימטריים בתהליכי העדכון השונים שעוברים על העץ.

9.6 עצים מאוזנים — (Balanced Trees)

בנית עץ חיפוש אופטימלי אינה משימה חד־פעמית. לאחר בניית העץ מגיעות רשומות חדשות או שיש לבטל רשומות קיימות. עדכונים אלו יכולים לגרום לכך שעץ אופטימלי יהפך לעץ לא אופטימלי. ניתן לומר שבדרך כלל עלות החיפוש בעץ חיפוש הולכת וגדלה ככל שמשך הפעילות גדל. כתוצאה מכך יש לבצע מזמן לזמן ארגון מחדש של העץ, כלומר לבנות מחדש את עץ החיפוש האופטימלי. לאחר בניית העץ מחדש יורדת עלות החיפוש והיא עולה בהדרגה ככל שחולף הזמן. עלות הארגון תלויה גם במספר הצמתים בעץ.

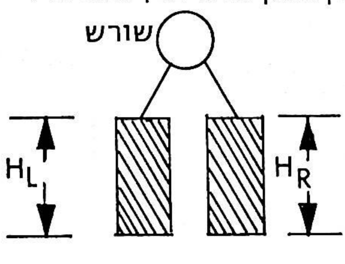


תרשים 9-24: עלות החיפוש כפונקציה של משך הפעילות בעץ בינרי

ארגון מחדש של העץ אפשר לבצע במועדים קבועים, או כאשר עלות החיפוש (זמן התגובה) עובר ערך מסויים קבוע מראש. שיטה זו אינה עונה על הדרישה לקיים זמן תגובה קבוע. ולכן פותחה שיטה לביצוע ארגון מחדש באופן דינמי. העצים המופקים בשיטה זו אינם עצים אופטימליים כי אם עצים מאוזנים.

בעץ מאוזן מתקיים עבור כל צומת התנאי הבא: ההפרש בין עומק תת־העץ הימני לבין עומק תת־העץ השמאלי אינו גדול מ־1.

עומק העץ מוגדר כמספר הקשתות במסלול הארוך ביותר בין השורש לצומת סופי. ניתן להוכיח כי עץ מאוזן קרוב מאד לאופטימום, ובמקרה הגרוע ביותר יש לבצע 44% יותר השוואות בעץ מאוזן מאשר בעץ אופטימלי.



$$|H_R - H_L| \leqslant 1$$

תרשים 9-25: עץ מאוזן

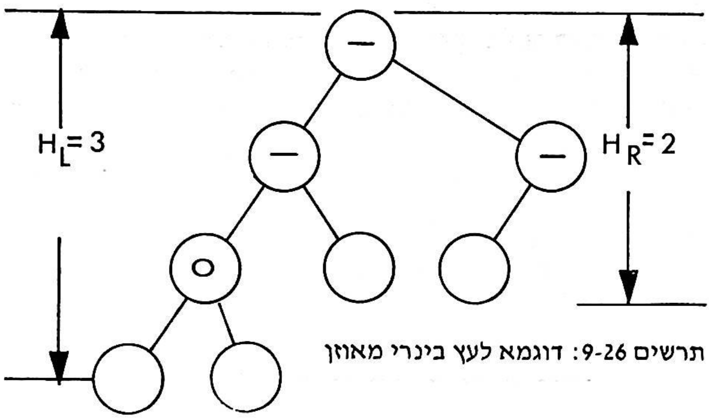
השמירה על העץ המאוזן נעשית באמצעות ארגון מחדש באופן שוטף כלומר, מיד כאשר מתגלה שהעץ אינו מאוזן. מספר פעולות התיקון שדרוש לבצע קטן יותר מאשר בעת ארגון מחדש ובניית עץ אופטימלי. נתאר את תהליך הארגון הדינמי בדוגמאות ובתרשימים.

בתרשימים 9-26, 9-27 נשתמש בסימונים הבאים לצמתים:

אם תת־עץ ימני עמוק יותר מהשמאלי נסמן את הצומת ב־+ ,

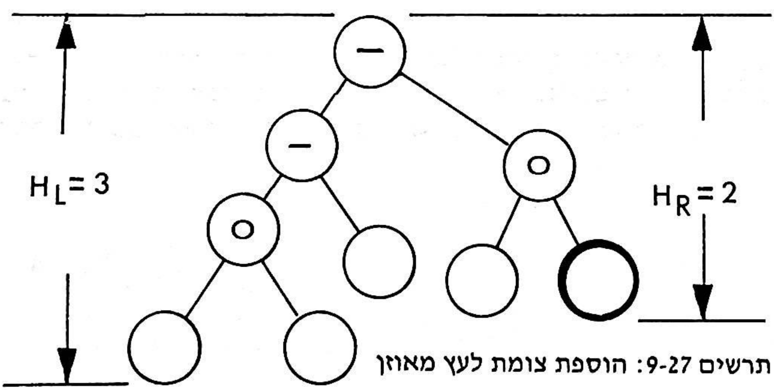
אם תת־עץ שמאלי עמוק יותר מהימני נסמן את הצומת ב־—,

אם שני תת־העצים בעלי עומק זהה נסמן את הצומת ב־0.



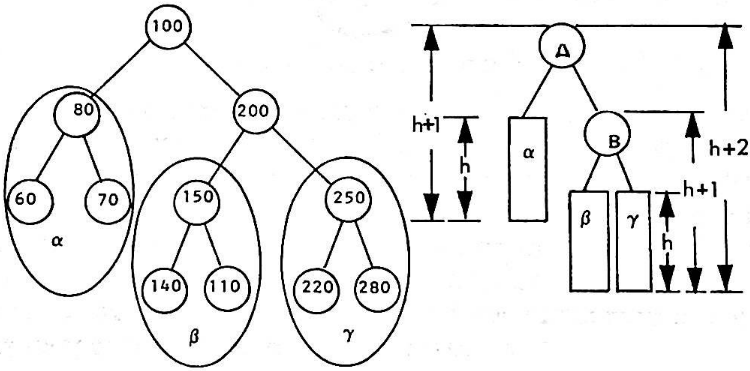
תרשים 9-26: דוגמא לעץ בינרי מאוזן

העץ בתרשים 9-26 הוא עץ מאוזן מאחר וההפרש בעומק בין כל תת־עץ ימני לשמאלי אינו עולה על 1. הוספת צומת חדש תשנה את הסימון של צומת אחד או יותר. בדוגמא שבתרשים 9-27 נוסף צומת חדש מימין. הסימון של צומת האב השתנה אך האיזון לא הופר. כאשר הצומת החדש מתווסף לתת־עץ העמוק ביותר הוא מפר את האיזון, ואז נדרש ארגון מחדש כפי שמודגם בתרשימים הבאים. בצד ימין של התרשימים יוצג המודל הכללי ובצד שמאל תוצג דוגמא מספרית.

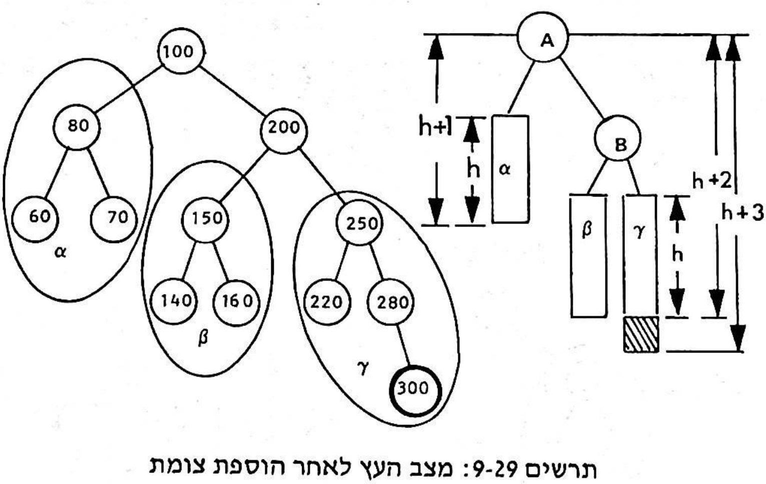


תרשים 9-27: הוספת צומת לעץ מאוזן

בתרשים 9-28 מוצג מצב קיים של עץ חיפוש בינרי. לעץ זה יש להוסיף צומת עם מפתח 300, ולפי מבנה העץ הוא יצטרף כבן לצומת 280. המצב החדש בתרשים 9-29 מציג עץ לא מאוזן. תת־עץ הימני של צומת A עמוק יותר ב־2 מאשר תת־עץ השמאלי שלו. יש לפעול עתה לתיקון המצב ולהחזרת איזון העץ.

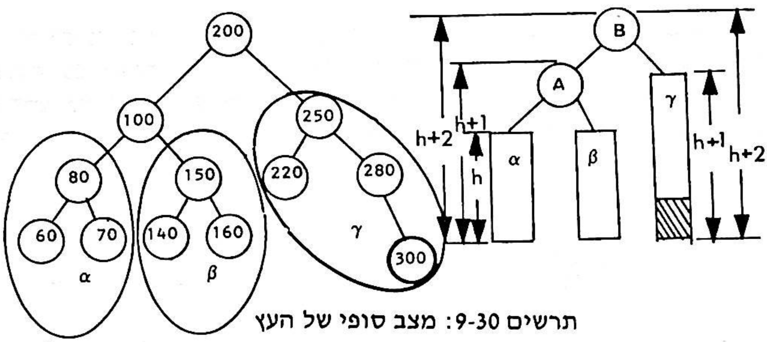


תרשים 9-28: עץ מאוזן (מצב לפני עדכון)



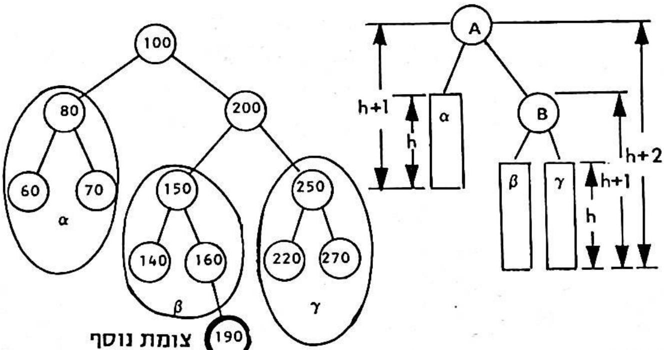
תרשים 9-29: מצב העץ לאחר הוספת צומת

אפשרות תיקון של איזון העץ המוצג בתרשים 9-29 תהיה על־ידי פעולת סיבוב שמאלה של כל הצמתים כדי שצומת B יהפוך להיות השורש וצומת A יהפוך להיות הבן השמאלי שלו. פעולה זו תגדיל את עומק תת־העץ השמאלי α ותביא לאיזון העץ כולו. תרשים 9-30 מציג את המצב החדש.

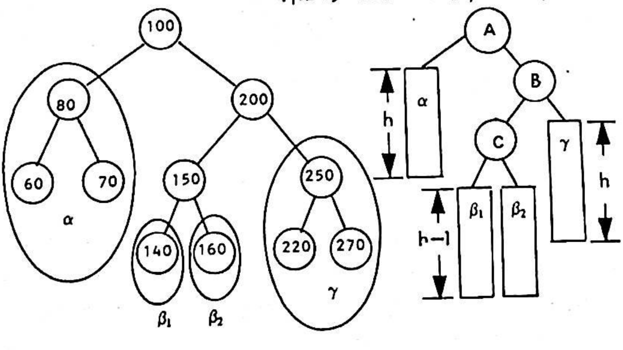


תרשים 9-30: מצב סופי של העץ

הפתרון שהוצג קודם התאים למצב שבו הצומת הנוסף מגיע לתת־עץ ימני קיצוני. כאשר הצומת החדש מגיע לתת־עץ שמאלי, כלומר לתת־עץ פנימי, דרושה פעולת תיקון שונה, כפי שמוצג בתרשימים הבאים. למצב ההתחלתי שבתרשים 9-31 יש להוסיף צומת חדש עם מפתח 190 אשר צריך להכנס לתת־עץ β. הוספת הצומת ל־ β תגדיל את העומק של β והוא יהיה עתה $h + 1$ ויגרום לכך שהעץ כולו לא יהיה מאוזן. תיקון של מצב זה מורכב יותר. תחילה יש לפצל את β לשני תת־עצים β_1 ו־ β_2 כפי שמודגם בתרשים 9-32. העץ מאוזן כי לא הוספנו עדיין את הצומת החדש.

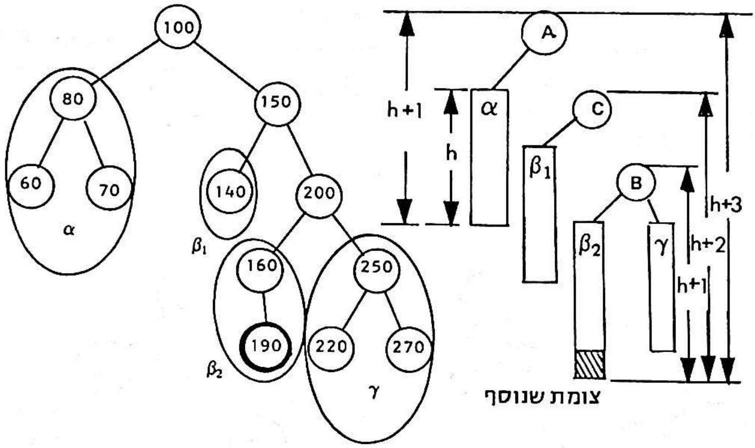


תרשים 9-31: עץ מאוזן (מצב לפני עדכון)



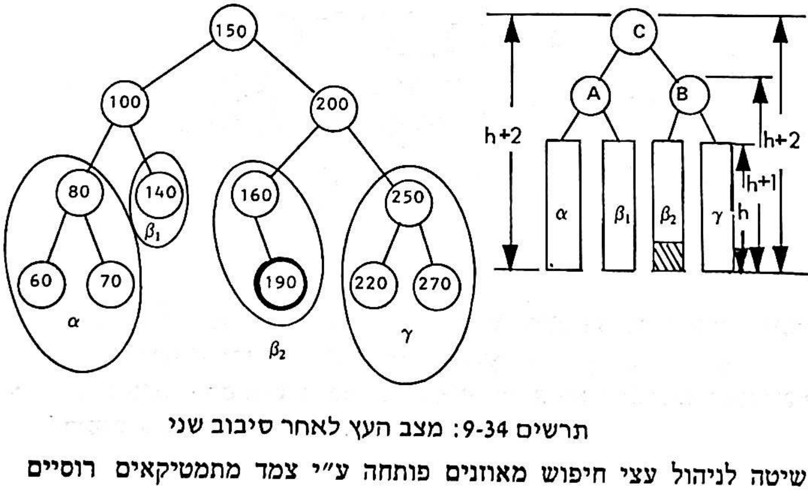
תרשים 9-32: מצב העץ לאחר פיצול

נוסיף עתה את הצומת החדש למקומו בתת־העץ הימני β_2 ונבצע פעולת סיבוב ימינה, כפי שמוצג בתרשים 9-33. בשלב זה העץ עדיין אינו מאוזן. יש לבצע עכשיו פעולת סיבוב שמאלה כדי לקבל עץ מאוזן.



תרשים 9-33: מצב העץ לאחר סיבוב אחד

העומק המקסימלי של העץ הסופי שבתרשים 9-34 לא השתנה. פעולת האיזון הינה פעולה מקומית בענף של העץ, ואין צורך לבצע ארגון מחדש של כל העץ. מצב זה לא קיים תמיד, מכיוון שלא ניתן להוסיף אינסוף צמתים חדשים ללא הגדלת עומק העץ. אם קיים צומת אחד לפחות לא מאוזן ניתן להוסיף צומת חדש ללא שינוי בעומק העץ. אם כל הצמתים מאוזנים, הוספת צומת חדש לא תפר את האיזון אבל תגדיל את העומק.

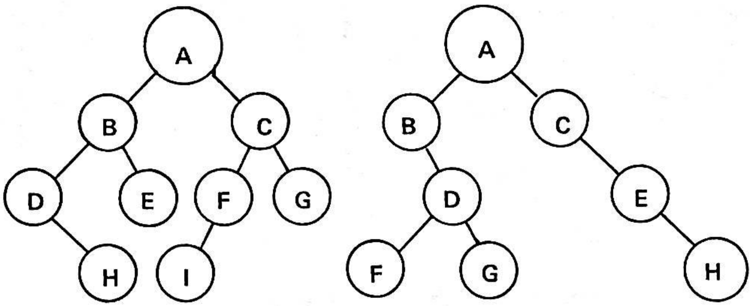


תרשים 9-34: מצב העץ לאחר סיבוב שני

השיטה לניהול עצי חיפוש מאוזנים פותחה ע"י צמד מתמטיקאים רוסיים Adelson-Velski ו־Landis ועל כן מקובל לקרוא לעצים עצי AVL.

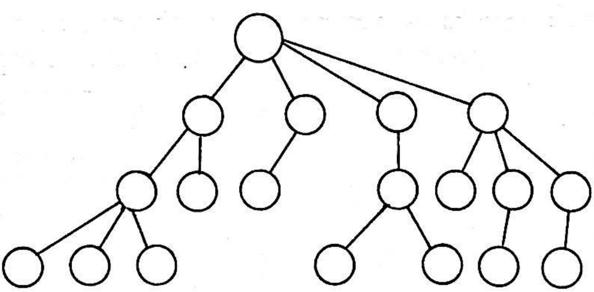
9.7 שאלות

1. קבע את סדר החיפוש בצמתים בעצים הבינריים A ו־B, לפי שלש השיטות שהוסברו.



עץ B   עץ A

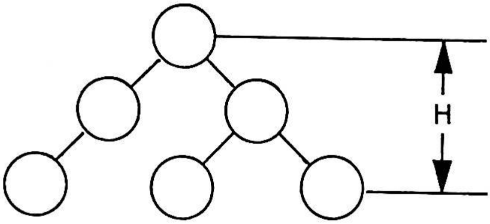
2. כתוב אלגוריתם לחיפוש In-Order בעץ בינרי.
3. בנה את העץ הבינרי המתאים לעץ הכללי:



4. הסבר את מושג עומק העץ ואת חשיבותו. חשב את עומק העץ הבינרי המכיל 50,000 רשומות. הסבר את תשובתך.
5. מדוע כדאי להשתמש במבנה עץ מאוזן. מדוע לא להשתמש במבנה עץ חיפוש אופטימלי.

9.8 נספח — חישוב עומק העץ

עומק העץ מוגדר כמספר המקסימלי של קשתות שיש לעבור כדי להגיע לצומת סופי ברמה הגבוהה ביותר. עומק העץ מציין את מספר ההשוואות והחיפושים שנצטרך לבצע על מנת להגיע לצומת היעד. הקשת היא הצורה הגרפית הנוצרת ע"י שני בנים הקשורים לאב אחד.



תרשים 9-35: מבנה עץ בינרי בעל עומק נתון

בהמשך נשתמש בסימונים הבאים:

- N — מספר הצמתים בעץ,
- M — מספר הקשתות בעץ,
- H — עומק העץ,
- m — מספר הבנים המקסימלי בצומת (דרגת העץ). בעץ בינרי $m = 2$.
- N_i — מספר הצמתים ברמה i,
- N_H — מספר הצמתים ברמה הנמוכה ביותר.

מאחר ומספר הבנים בכל צומת קטן או שווה ל־m הרי שמתקיים $N_1 \leqslant m$, $N_2 \leqslant m^2$ או באופן כללי: $N_i \leqslant m^i$, וזאת מאחר ולא כל הבנים קיימים.

$$N = \sum_{i=0}^{H} N_i \leqslant \sum_{i=0}^{H} m^i = \frac{m^{H+1} - 1}{m - 1}$$

$$H = \log_m [N(m-1) + 1] - 1$$

עבור $N > 1$ נקבל בקירוב:

$$H \cong \log_m N$$

פרק עשירי
תכנון קבצים

10.1 מבוא
10.2 מודל כללי לתכנון קבצים
10.3 מאפייני קבצים
10.4 סיכום
10.5 אירוע — תכנון קבצים
10.6 שאלות

בפרק זה נציג את המאפיינים השונים של הקבצים והשיקולים השונים בתכנונם.

10.1 מבוא

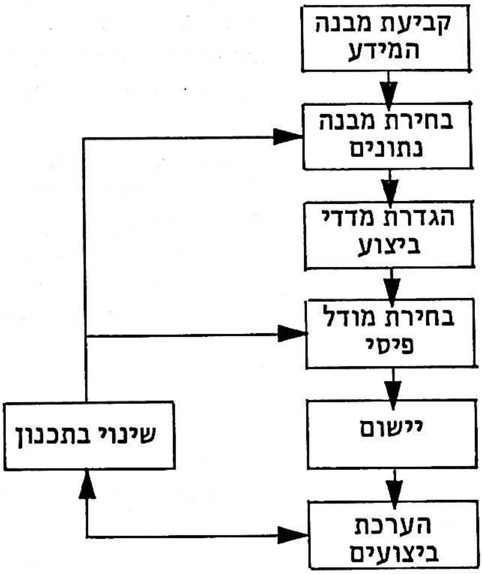
תכנון הקובץ קובע את **שיטת איתור הרשומות** ואת שיטת העדכון. התכנון צריך להתחשב במבנה הנתונים והקשרים ביניהם, סוגי הציוד וסוג מערכת התוכנה לניהול קבצים העומדים לרשות המתכנן. תכנון הקובץ הינו בעל חשיבות רבה מאחר והוא משפיע על הביצועים של מערכת המידע.

החיפוש של רשומות הינו בדרך כלל החלק הצורך את מירב הזמן בתוכניות מחשב. החלפת שיטת ארגון קובץ גרועה בשיטת ארגון קובץ טובה, מבחינת החיפוש, יכולה להביא לשיפור ניכר בביצועי התוכנית.

נציג מודל כללי לתכנון קובץ ואת המאפיינים השונים של הקבצים. מאפייני הקובץ חשובים לבחירת מבנה הנתונים המיועד לפתור את בעית עיבוד הנתונים. נציין שאין טכניקה "טובה" לכל המצבים. ההחלטה על שימוש בטכניקה מסוימת צריכה להביא בחשבון את הדרישות ממערכת המידע. יעילותה של טכניקת ארגון נתונים תלויה ביישום ובפונקצית המטרה הכוללת בין השאר את זמן האיחזור וזמן העיבוד לעדכון הקובץ.

10.2 מודל כללי לתכנון קבצים

תהליך תכנון קובץ מורכב ממספר שלבים כפי שמוצג בתרשים 10-1. בתהליך זה קיימת מערכת היזון חוזר המופעלת לאחר שבשלב בחינת יישום המערכת מתגלות סטיות מן הדרישות הפונקציונליות, ממצבי הביצוע, או מהתכנון עצמו. תהליך התיקון צריך לגרום לבחינה חוזרת של הדרישות, המדדים והתכנון כדי להפיק מוצר סופי טוב יותר.



תרשים 10-1: הצגה גרפית למודל כללי לתכנון קבצים

א. קביעת מבנה מערכת המידע (Information Structure Selection)

בשלב זה מנתח המערכות מגדיר את הרכיבים הבונים את מערכת המידע ומציג את תוכנם ואת הקשרים ביניהם כפי שהם נלמדו ונחקרו בתהליך ניתוח ותכנון המערכת. את הרכיבים האלה יש להציג במערכת הקבצים ולנהל במערכת המידע הממוחשבת. לדוגמא, במערכת לניהול מלאי הרכיבים המרכזיים הם מידע על הפריט, מידע על הכמות במלאי, מידע על ההזמנות, מידע על ספקים ולקוחות ועוד. קשרים קיימים בין פעילויות ברכיבים שונים, למשל כניסה מהזמנה מקטינה את הכמות בהזמנה ומגדילה את הכמות במלאי, או ירידה במלאי מתחת לערך מסויים גורמת להוצאת הזמנה לחידוש מלאי.

ב. בחירת מבנה נתונים (Data Structure Selection)

בשלב זה יש לארגן את כל הרכיבים המשתתפים למבנה נתונים לוגי של קבצים, רשומות ושדות ולבחון את אפשרויות הגישה לנתונים. לא קיימת שיטה מוגדרת לבחירת מבנה נתונים ולעיתים ניתן לפתור את אותה בעית עיבוד נתונים על־ידי שימוש במספר מבני נתונים שונים. למידת הנסיון המקצועי של מנתח המערכות חשיבות רבה בשלב הזה.

ג. הגדרת מדדי ביצוע (Performance Requirements)

על סמך הדרישות ממערכת המידע יש לקבוע מספר מדדי ביצוע. מדדי ביצוע אופייניים יכולים להיות זמן גישה ממוצע לשליפת רשומה, זמן עדכון רשומה וכדומה. את מדדי הביצוע ניתן להגדיר בעזרת נתונים כגון — מספר ממוצע של רשומות שיש לשלוף לשם תשובה לשאילתה, תדירות השאילתות, תדירות העדכונים, אפיון הסיבוך של השאילתה הממוצעת, סוג הציוד בו משתמשים ועוד.

ד. בחירת מודל פיסי לאחסון (Storage Structure Selection)

בשלב זה יש להתאים את המודל הלוגי של הנתונים למודל פיסי כלשהו המוכר ומנוהל על־ידי המערכת לניהול קבצים הקיימת במחשב שבו תבוצע העבודה. בשלב זה כבר ידועים נתונים כגון: אורך רשומה (מקסימלי, מינימלי וממוצע), מספר רשומות בקובץ, מספר מפתחות, סוג המפתחות, סוגי העדכון שיש לבצע. מאחר וקיימות מספר רב של מערכות לניהול קבצים, ניתן להפוך את אותו מבנה לוגי למספר מבנים פיסיים, כל אחד בעל יתרונות וחסרונות משלו. יתכן שבשלב זה יהיה צורך לבצע מספר שינויים במודל הלוגי כדי להתאימו למערכת התוכנה לניהול קבצים.

בגמר תהליך בחירת מבנה הנתונים לאחסון ידועים למתכנן מספר מאפיינים חשובים כגון — נפח האחסון הדרוש ביחידות האחסון, זמן ממוצע לשאילתה, מספר ממוצע של גושים שיש לקרוא כדי לענות לשאילתה (גושי נתונים או גושי אינדקסים), ועוד.

ה. יישום (Implementation)

יישום המודל הפיסי. הקמת הקבצים והפעלת מערכת המידע.

ו. הערכת ביצועים (Performance Evaluation)

על פי מדדי הביצוע שנקבעו יש לבדוק אם המבנה הפיסי שנבחר אכן עומד בציפיות. במידה ולא, יתכן שיהיה צורך לבצע מספר שינויים במודל הלוגי או במודל הפיסי, או בשניהם. שינויים אפשריים הם: הוספת מפתחות לגישה, הוספה או ביטול קשרים בין נתונים, שינוי במבנה הרשומות (גודל וחלוקה פנימית), הוספת אינדקסים או ביטול ועוד.

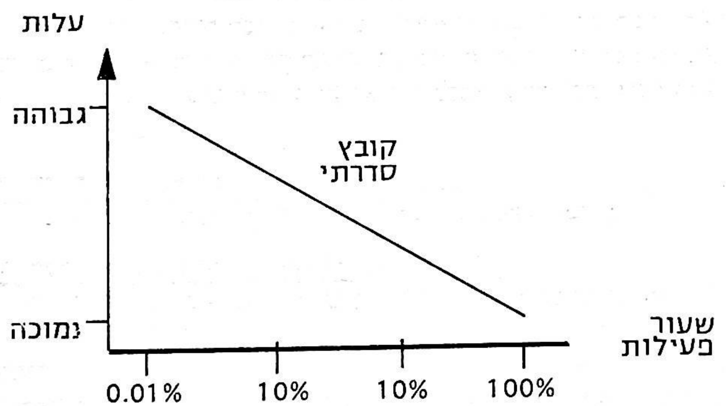
10.3 מאפייני קבצים

המאפיינים השונים של הקבצים וההיבטים הכלכליים הקשורים בשיטות הארגון השונות הינם רבים. נסקור את החשובים שבהם.

א. שעור פעילות הקובץ

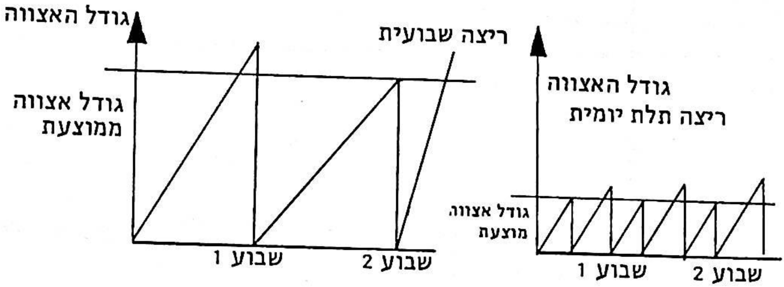
שעור הפעילות (Activity Rate) מוגדר כיחס בין מספר הרשומות הפעילות בזמן העיבוד לבין המספר הכולל של הרשומות בקובץ. רשומה פעילה מוגדרת כרשומה הנשלפת בזמן העיבוד. לדוגמא, שליפת 10,000 רשומות מקובץ המכיל 100,000 רשומות מייצג שעור פעילות של 10%. לעומת זאת שאילתא השולפת רשומה בודדת מאותו הקובץ מייצגת שעור פעילות של 0.001%.

בארגון סדרתי מעבדים תמיד את כל הרשומות, ללא תלות במספר רשומות התנועה. למרות זאת נגדיר כרשומות פעילות רק את אותן רשומות מקובץ האב שעבורן קיימת תנועה. בארגון סדרתי, עיבוד כל הקובץ או עיבוד חלק ממנו מתבטא בהפרש זמן קטן (הפרש זה מותנה כמובן בסוג העיבוד המבוצע ברשומות הפעילות) מאחר וצוואר הבקבוק נוצר מסיבות קלט/פלט ולא ממשך העיבוד. זמן המחשב הנוסף הדרוש לעיבוד כתוצאה מתוספת תנועות הוא קטן בדרך כלל. ניתן לומר שהעלות לעיבוד תנועה, אשר מתבטאת במשך העיבוד, יורדת ככל ששעור הפעילות גדל. יחס זה הוא יחס לוגריתמי.



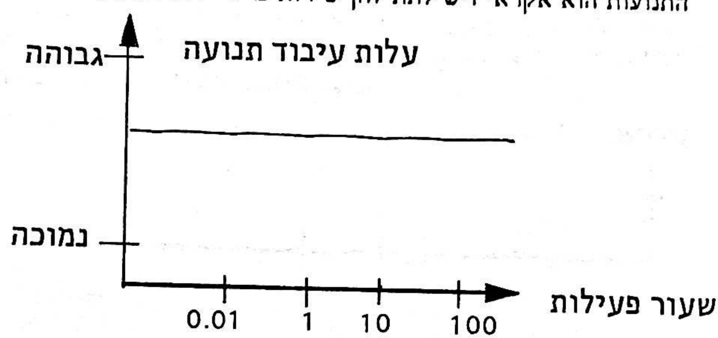
תרשים 10-2 : עלות עיבוד כפונקציה של שעור פעילות בקובץ סדרתי

ניתן לראות שאין זה כלכלי לעבד קובץ בארגון סדרתי כאשר שעור הפעילות הוא נמוך. כדי להגדיל את יעילות העיבוד הסדרתי, ניתן להגדיל את שעור הפעילות על־ידי צבירת תנועות ועיבוד באצווה (Batching), כלומר יש לבצע עדכון תקופתי. צבירת תנועות וריצה (Run) פעם בשבוע, תהיה יותר יעילה מבחינת עלות העיבוד, מאחר וגודל האצווה הממוצעת גדול יותר.



תרשים 10-3: גודל האצווה כפונקציה של מספר הריצות התקופתיות

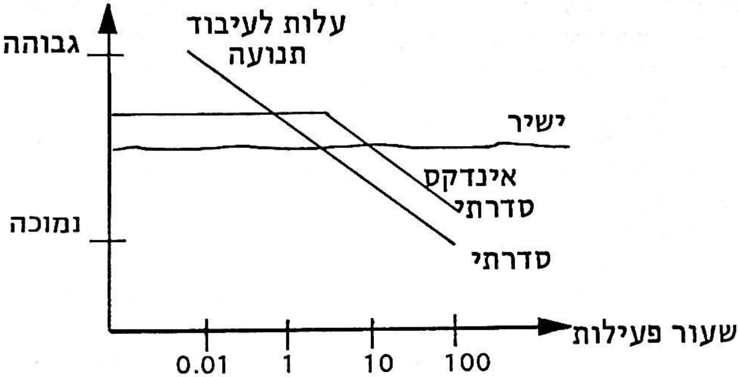
בארגון לגישה ישירה, הרשומות ניתנות לשליפה מיידית. זמן עיבוד הרשומה נשאר קבוע, ללא תלות בגודל האצווה, מאחר ובכל מקרה שולפים רק את הרשומות הדרושות. בארגון ישיר אין שום יתרון לבנית אצוות, ומסיבה זו נדרשת שיטת העיבוד בגישה ישירה במערכות תקשורת שבהן מופע התנועות הוא אקראי ויש לתת להן שירות מיידי ללא צבירה באצוות.



תרשים 10-4: עלות עיבוד כפונקציה של שעור הפעילות בארגון ישיר

בארגון האינדקס סדרתי יש יתרון לשעור פעילות גבוה מאחר והוא מאפשר עיבוד סדרתי גם ללא האינדקסים. עם זאת הוא יקר יותר מהארגון הסדרתי מאחר ועליו לעדכן את האינדקסים בעת הוספת וביטול של רשומות. בשעור פעילות נמוך, ארגון האינדקס הסדרתי יקר יותר מהארגון הישיר בגלל הצורך לטפל באינדקסים. למרות העובדה שיטת האינדקס הסידרתי יקרה יותר, בכל שעור פעילות, משתי השיטות הקודמות. היתרון הגדול שלה הוא השילוב בין

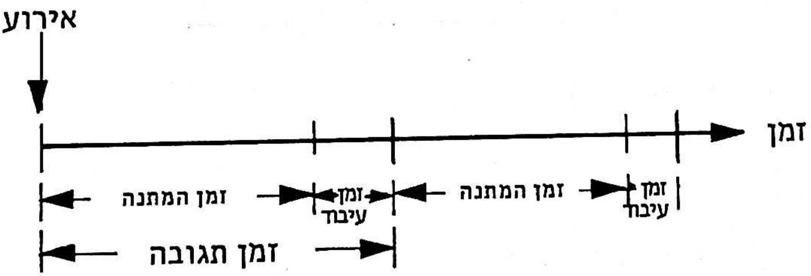
שתי צורות גישה שונות לרשומות והאפשרות של המשתמש לבחור לפי התנאים בשיטה היעילה יותר.



תרשים 10-5: עלות עיבוד תנועה כפונקציה של שעור פעילות ושיטות ארגון הקובץ

ב. זמן תגובה

זמן התגובה (Response Time) מוגדר כמשך הזמן החולף מיצירת אירוע ועד לייצוג האירוע במערכת המידע. זמן התגובה כולל את זמן ההמתנה עד לביצוע העיבוד ואת זמן העיבוד עצמו (תרשים 10-6). באופן כללי, ניתן לומר שכדי להשיג זמן תגובה קצר יותר יש להעלות את עלות מערכת המידע כדי להפיק את אותו עיבוד. מאידך, התועלת המופקת ממידע מעודכן מצדיקה ולעיתים אף מחייבת את התוספת בעלות העיבוד.



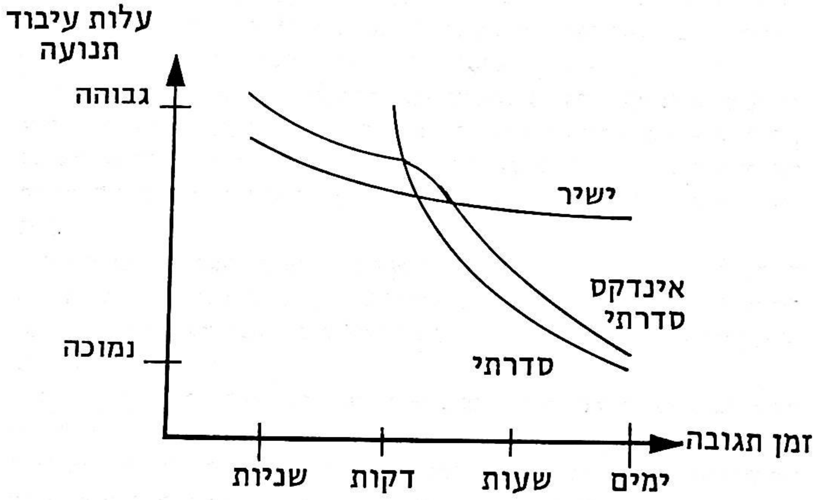
תרשים 10-6: זמן תגובה

קיימים יישומים בהם עיבוד תקופתי מספיק. צבירת תנועות עדיפה כאשר התועלת המופקת ממידע עדכני קטנה מעלות העיבוד. הצבירה יכולה להמשך שעות וימים. בארגון סדרתי זמן התגובה יכול לנוע בתחום בלתי מוגבל ובלתי מגודר מראש. אך ככל שזמן התגובה הדרוש קטן יותר, עלות העיבוד תהיה

גדולה יותר מאחר והאצוות תהיינה קטנות יותר וידרשו מחזורי עדכון רבים יותר להשלמת עדכון התנועות.

הארגון הישיר מאפשר זמן תגובה מהיר ביותר ואין יתרון לבנית האצוות או מיון התנועות לפני העיבוד. באופן כללי ניתן לומר שעלות העיבוד בשיטה ישירה היא קבועה ביחס לזמן התגובה הדרוש.

ארגון האינדקס הסדרתי נמצא בין שתי שיטות הארגון הקודמות. ככל שזמן התגובה הדרוש גדול יותר, עלות העיבוד תהיה נמוכה יותר בגלל היעילות שבבנית אצוות ומיון התנועות. בזמני תגובה קצרים, השיטה יקרה יותר מאשר בארגון הישיר בגלל הצורך בסריקת אינדקסים.



תרשים 10-7: עלות עיבוד תנועה כפונקציה של זמן התגובה ושיטת ארגון הקובץ

ג. סוג היישום

קיימים יישומים המתבצעים ביעילות רבה בארגון סדרתי בקבצים הממוינים בסדר מסוים. למשל בחישובי משכורת, יש לסרוק את כל הקובץ פעם בתקופה ולחשב לכל עובד את המשכורת. לעומת סוג זה של ישומים, קיימים יישומים המחייבים גישה ישירה לנתונים. למשל במערכת להזמנת כרטיסי טיסה, יש לאפשר גישה ועדכון מיידי לכל הזמנת כרטיס ואין כל אפשרות לתת תשובה מיידית ללקוח בשיטות עיבוד באצווה. כל יישום בו נדרש עדכון ואיחזור נתונים מיידי מחייב אחזקת הקבצים ביחידות אחסון בגישה ישירה. קיימים יישומים בהם יש צורך בגישה ישירה לקובץ וגם גישה סדרתית, ולכן משתמשים עבורם בארגון האינדקס הסדרתי.

יישום נמדד גם במידת האינטגרציה בין הקבצים השונים. אירוע מסוים יכול להיות קשור בקבצים שונים. לדוגמא, מכירת מוצר מחייבת עדכון רשומת

לקוח ורשומת מלאי. בארגון סדרתי האינטגרציה מבוצעת על־ידי מספר ריצות עדכון שונות. בארגון נתונים אחר ניתן לבצע עדכון בו־זמני של מספר קבצים בשיטות של גישה ישירה. ביישומים שבהם יש אינטגרציה בין הקבצים דרוש תכנון מדוקדק של כל הקבצים כדי לאפשר עדכון בו־זמני.

ד. גודל הקובץ

הדיסקים שהם יחידות לאחסון לגישה ישירה (DASD—Direct Access Storage Devices) יקרים יחסים לסרטים מגנטיים ולכן מחיר אחסון יחידת מידע בהם גבוה יותר. אחזקת קובץ ביחידת אחסון לגישה ישירה יקרה יותר בהשוואה לאחזקתו בסרט מגנטי. אם הקובץ גדול מאד, אחזקתו על דיסק פירושה תפיסת מספר רב של כוננים. לעיתים מעבירים את הקובץ מסרט לדיסק לצורך עיבודו, ובגמר העיבוד מחזירים אותו לסרט.

הדיון שלמעלה **טוב** רק למקרה שבו עומדת ברירה לפני המשתמש בין אחזקת הקובץ בדיסק מגנטי או בסרט, או כאשר הוא זקוק לקובץ לשם עדכון בגישה ישירה בלבד ולא למשך עיבוד שוטף. גם במקרה זה יש לשקול את העלות של העברת הקובץ מסרט לדיסק ולהיפך לעומת עלות אחזקתו בדיסק בלבד.

עם התפתחות מערכות עיבוד בתקשורת אין למעשה למשתמש כל ברירה שהיא. החלטתו להפעיל מערכת עיבוד בתקשורת גורמת **הפעלת יחידות אחסון לגישה ישירה בלבד** לצורך גישה ישירה לקבצים כל זמן שמערכת התקשורת משרתת את הצרכנים.

לגודל הקובץ חשיבות רבה. בארגון אינדקס סדרתי גודל הקובץ משפיע על מספר רמות האינדקס ולכן על זמן הגישה לרשומה. בארגון ישיר אין "דוחסים" את הקובץ כדי לאפשר טיפול נוח בגלישות, ופירושו השארת שטחי אחסון פנויים. המשמעות הינה הגדלת נפח האחסון הדרוש.

ה. שעור גידול הקובץ

שעור גידול הקובץ הוא היחס בין נפחַ הקובץ בין שתי יחידות זמן אשר מושפע ממספר הרשומות המתווספות והמתבטלות במשך התקופה. לדוגמא, קובץ תנועות מלאי הוא קובץ הגדל כל הזמן מאחר ומצטרפות אליו כל תנועות המלאי של העיבוד האחרון. לעומתו קובץ המלאי הוא קובץ הגדל פחות, בשיעור של מספרים קטלוגיים חדשים ומספרים שמתבטלים.

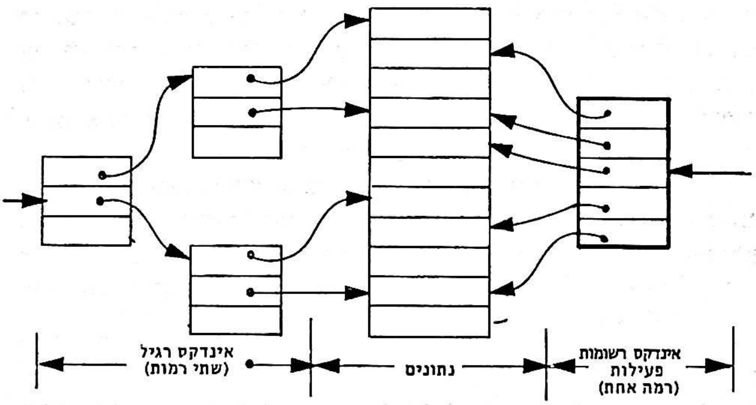
שיעור הגידול של הקובץ משפיע על עלות העיבוד, מאחר וכל שיטה משתמשת בטכניקות שונות לטיפול ברשומות החדשות או לביטול רשומות. בארגון סדרתי, הוספת רשומה או ביטולה מחייבת העתקת הקובץ לקובץ חדש. פעולות אלו הן חלק מתהליך העדכון שבו מעתיקים **תמיד** את הקובץ.

בארגון ישיר או אינדקס סדרתי קיים קושי מסוים בהוספת או ביטול רשומות בגלל בעית הגלישות. לאחר שעור גידול מסוים, שתי השיטות מחייבות ארגון מחדש של הקובץ. גידול מהיר של קובץ, יכול לחייב יחידות אחסון נוספות ויכול להשפיע על זמן העיבוד בגלל העברת רשומות לשטחי גלישה.

קיימות שיטות ניהול קבצים שאינן מאפשרות לקובץ לגדול מעבר לגודל שנקבע בזמן בנייתו. תיתכן שיטה המתאימה לקובץ בגודל מסויים ואינה מתאימה לקובץ השונה ממנו בגודלו.

ו. תדירות פניה לרשומה

תדירות הפנייה לרשומות הקובץ אינה שווה. יש רשומות שפונים אליהם בתדירות גבוהה ואילו לאחרות הפניות הן בתדירות נמוכה יחסית. לדוגמא מערכת הזמנת כרטיסי טיסה שבה הולך וגדל מספר הפניות לרשומות המייצגות את הטיסה ככל שמועד הטיסה מתקרב. לאחר ביצוע הטיסה, תדירות הפניה לרשומות יורדת באופן משמעותי ולמעשה אין פניות כלל. דוגמא אחרת היא של חשבונות עובר ושב בבנק שבו קיימים חשבונות פעילים מאד וחשבונות פעילים פחות. על המתכנן לשקול כיצד לייעל את הגישה לרשומות הפעילות מאד מבלי להעלות את מחיר המערכת עבור כל הרשומות. ניתן להחזיק אותן בזכרון או לבנות אינדקס מיוחד בעל מספר רמות קטן יותר לרשומות הפעילות בלבד כפי שמוצג בתרשים 10-8.



תרשים 10-8: קובץ עם אינדקס רגיל ואינדקס לרשומות פעילות

ז. גודל הרשומה

שיטות הארגון תומכות ברשומות במבנים שונים, ולא תמיד בכל המבנים האפשריים. אם חייבים להשתמש ברשומות באורך משתנה, למשל, אשר אינם מקבלים תמיכה בשיטת ארגון ובתוכנה מסויימת, יש צורך לחפש דרך שונה להגדרת הרשומות או לעבור לשיטת ארגון אחרת.

ח. שעור הדחיסות

כאשר כל השטח המוקדש לקובץ מלא ברשומות נאמר שהקובץ דחוס ב־100%. קיימות שיטות בהן מתבצעת הקצאה מראש של כל שטח הדיסק שידרש לקובץ למרות שמספר הרשומות הנמצאות בנקודת זמן זו בקובץ קטן בהרבה ומתקבל קובץ לא־דחוס. כאשר הקובץ אינו דחוס מתעוררת הבעיה של אפשרות האבחנה בין מקום תפוס למקום ריק. קיימות מערכות לניהול קבצים המטילות אחריות זו על המשתמש ועליו לזהות אם הרשומה קיימת או לא. במערכות מתקדמות יותר, נושא זה מנוהל על־ידי התוכנה לניהול קבצים.

בעיה נוספת בקבצים לא צפופים היא ניצול השטח הפנוי מחדש. קיימות שיטות בהן אין אפשרות להשתמש בשטח מחדש, בעוד אחרות מנצלות את השטח שהתפנה לאחסון רשומות חדשות מבלי להגדיל את הקובץ. בנושאים אלו דנו בפרקים קודמים. ארגון נתונים בגישה ישירה רגיש מאד לצפיפות הקובץ מפאת הצורך בניהול הרשומות הגולשות.

ט. גיבוי

בכל מערכת מידע ממוחשבת קיים אלמנט של גיבוי (Backup) המאפשרת התאוששות וחזרה לעבודה תקינה לאחר תקלה במערכת. הארגון הסדרתי מאפשר גיבוי פשוט יחסית מאחר והוא אינו פוגע ברשומות הקיימות (קובץ אב) לצורך העדכון, ובכל מעבר עדכון מופק קובץ בן חדש. במקרה של תקלה, ניתן להריץ פעם נוספת את אותה תוכנית ולהפיק קובץ־בן חדש. בשיטות ארגון אחרות עדכון רשומה בקובץ מתבצע על ידי עדכונה בזכרון והחזרתה מעודכנת ליחידת האחסון במקום הרשומה המקורית. בשיטות האלה, תקלה בעדכון אינה מאפשרת התאוששות מיידית מאחר ויש צורך להחזיר תחילה את הקובץ למצבו המקורי לפני העדכון. מערכות המשתמשות בשיטות ארגון מתקדמות מחייבות מערכי גיבוי מורכבים כדי לאפשר התאוששות לאחר תקלה וחזרה לעבודה תקינה.

י. שימוש צפוי

קיימים קבצים המשמשים לקריאה בלבד (Read Only) כמו למשל קובץ יתרות חשבון המשמש לקריאה בכל עת ומתעדכן אחת לתקופה. אם נבנה אינדקסים משניים רבים לקבצים אלה נוכל לקבל זמני גישה סבירים לפי מספר מפתחות מבלי להעלות את זמן העדכון. בקבצים אלה ניתן לבנות את האינדקסים אחת לתקופה, למשל לאחר העדכון, מאחר ואין הם משתנים בין התקופות. לעומתם בקבצים עם עדכונים תכופים כל מפתח משני נוסף מגדיל בצורה ניכרת את זמן העדכון.

יא. מניעת גישה בלתי מורשית

במערכות עיבוד שונות ובמיוחד במערכות לגישה ישירה הפועלות בתקשורת יש לקבוע את ההרשאה לגישה מבחינת הסודיות או הבטיחות במערכת. יש לקבוע כללים אילו תוכניות ואילו משתמשים יכולים לגשת לקבצים או לרשומות ואילו פעולות מותרות לביצוע על ידי כל משתמש (קריאה בלבד, ביטול, עדכון).

יב. אמינות יחידות אחסון

הקבצים מאוחסנים ביחידות אחסון שונות ומורכבים בכוננים הקשורים ליע"מ באמצעות ערוצים לשם עיבוד. תקלה יכולה לקרות בכל אחת מהחוליות ולמנוע את זמינות הקובץ למשתמש. קיימות שיטות שונות להעלאת זמינות הקבצים כמו למשל על ידי אחזקת כונני דיסקים חליפיים אחזקת קובץ כפול על כוננים שונים בערוצים שונים, ועוד.

10.4 סיכום

נסכם בקצרה את השיקולים השונים לבחירת קבצים. בחירת מצע האחסון מכתיב את שיטת הארגון. אחסון הקובץ בסרט מאפשר שיטת ארגון סדרתית בלבד. השיטות המתקדמות אפשריות רק ביחידות אחסון בגישה ישירה. השיטה הסדרתית דורשת עיבוד כל הקובץ ולכן בהשוואה לשיטות האחרון היא אינה יעילה בשעורי פעילות נמוכים. במציאות נמצא שבקבצים רבים שעור הפעילות הוא פחות מ־10%.

שיטות הארגון המתקדמות מאפשרות גישה לרשומה לפי מפתח. זמן נוסף דרוש לתרגום המפתח לכתובת הפיסית של הרשומה. שיטות אלה משנות את תוכן הקובץ בזמן העיבוד ולכן מחייבות מנגנונים מיוחדים לשחזור והתאוששות. בשיטה הסדרתית נוצר קובץ חדש בכל עדכון ולכן הקובץ הישן נשמר כפי שהיה לפני העיבוד.

יישום המחייב עדכון מיידי בי זמני של מספר קבצים ניתן לביצוע רק בשיטות ארגון מתקדמות. בארגון סדרתי, עדכון של מספר קבצים מחייב מספר מעברי עדכון נפרדים.

הקריטריונים העקריים לבחירת שיטות ארגון לא סדרתיות הם:

— **מהירות גישה:** הארגון הישיר הוא המהיר ביותר.
— **דרישות עיבוד:** אינדקס סדרתי מאפשר גם עיבוד סדרתי וגם עיבוד ישיר. בארגון הישיר אין אפשרות לעיבוד סדרתי אלא אם מבצעים מיון.
— **שעור דחיסות הקובץ:** הארגון הישיר הוא הגרוע ביותר מבחינה זו, כלומר יש ניצול נמוך של שטח האחסון.
— **ארגון מחדש של הקובץ:** נדרש במערכות שבהן נוצרת גלישת רשומות.

10.5 אירוע — תכנון קבצים

מטרת האירוע היא להציג יישום של המודל לתכנון קבצים שהוצג בסעיף 10.2 ולהדגיש השיקולים השונים המנחים את מנתח המערכות בתכנון רשומות ושיטות לארגון קבצים. הפתרון המוצג מתרכז רק בנושא ארגון קבצים ותכנון רשומות ואינו עוסק בניתוח מערכת המידע. יש לציין שאין זה הפתרון היחיד או היעיל ביותר דוקא. הוא נבחר בעיקר בגלל מגוון השיטות לארגון קבצים המופיעות בו. האירוע אינו מכוון לפתור בעיה בשלימותה אלא להציג את דרך הפתרון בלבד ולכן אין להקיש ממנו למציאות כלשהי.

תיאור הבעיה

אוניברסיטה גדולה החליטה למכן את הספריות בקמפוס. הספריות מפוזרות מבחינה גיאוגרפית בכל רחבי הקמפוס ומכילות כרבע מיליון ספרים. הספריות מנהלות באופן נפרד את ההשאלות. באוניברסיטה כ־20,000 סטודנטים המבצעים כ־400,000 השאלות במשך השנה.

יש לבנות מערכת מידע שתשרת את כל הספריות ותאפשר:

א. הכנסת כל הנתונים למערכת בצורה ישירה באמצעות מסופים,

ב. גישה מהירה למידע אודות ספרים, סופרים, השאלות וכדומה,

ג. הפקת מספר דוחות תפעוליים.

רשימת התנועות השונות להכנסת נתונים למערכת:

1. הוספת ספר חדש. התנועה תכלול נתונים על הסופר או סופרים, פרטים כלליים של הספר ומילות המפתח של הספר. על המערכת לבדוק אם הסופרים ומילות המפתח מוגדרים במערכת ואם לא, להתריע על כך.
2. עדכון נתוני ספר. שינוי שם סופר או מילת מפתח מחייב בדיקה אם הם מוגדרים במערכת.
3. ביטול ספר. הביטול יגרום לביטול כל המידע הקשור בספר. הביטול אפשרי רק אם הספר אינו נמצא באף ספריה ואין לו השאלות.
4. הוספת מילות מפתח לספר או ביטול מילות מפתח.
5. הוספת סופר חדש למערכת. התנועה תכלול פרטים כלליים על הסופר.
6. עדכון נתוני סופר.
7. ביטול רשומת סופר. המערכת צריכה לבדוק שלסופר זה אין אף ספר בספריה.
8. הוספת ספריה חדשה למערכת. התנועה תכלול פרטים כלליים על הספריה (למשל, קבוצות הספרים שנמצאים בה).
9. עדכון נתוני ספריה.
10. ביטול רשומת ספריה. המערכת תבדוק שאין ספר הרשום בספריה זו.

11. הוספת עותק של ספר לספריה. המערכת תבדוק אם הספר והספריה מוגדרים.
12. ביטול עותק מספריה.
13. פתיחת השאלה. התנועה תכלול את הנתונים הכללים של הסטודנט ואת נתוני הספר המושאל.
14. החזרת ספר לספריה.

המערכת צריכה לאפשר שאילתות בגישה ישירה ומספר מסוים של דוחות תפעוליים. רשימת השאילתות בגישה ישירה:

1. מירב הפרטים על ספר. גישה לפי שם הספר.
2. מירב הפרטים על סופר. גישה לפי שם סופר.
3. רשימת הספרים שנכתבו על־ידי סופר מסוים. גישה לפי שם סופר.
4. רשימת ספרים בנושא מסוים. גישה לפי מילת מפתח אחת או יותר.
5. מספר העותקים של ספר נתון בספריה מסויימת. גישה לפי סמל ספריה ושם ספר.
6. רשימת ההשאלות לספר בספריה נתונה. גישה לפי סמל ספריה ושם ספר.
7. רשימת הספרים שהושאלו לסטודנט. גישה לפי מספר תעודת זהות וסמל ספריה.

הדו"חות התפעוליים במערכת:

1. רשימת הסופרים ממוין לפי א׳-ב׳.
2. רשימת הספרים לפי נושא.
3. רשימת הספרים שעבר שבוע מאז תאריך החזרה צפוי. מיון לפי סמל ספריה ושם ספר.

במערכת מסוג זה אפשריים תנועות, שאילתות ודו"חות נוספים המותאמים לנוהלי העבודה, לשיטות הארגון ועוד.

תיאור הפתרון

א. זיהוי המרכיבים של הבעיה

בשלב זה נזהה את רכיבי הנתונים הנדרשים במערכת לניהול ספריות.

— עבור כל ספר יש לנהל את הנתונים הבאים:

* נתונים כלליים על הספר (שם הספר, מספר קטלוגי, שפה, שנת הוצאה).
* מילות מפתח. לכל ספר יתכן מספר רב של מילות מפתח המזהות את נושא הספר.
* המחברים. לספר יתכן מחבר אחד או יותר.
* מלאי הספר. הספר יכול להופיע במספר ספריות במספר עותקים. עבור כל ספריה יש לנהל נתונים על כמות עותקים תקנית, כמות עותקים בפועל, תאריך כניסת הספר לספריה.
* השאלות. לכל ספר יתכנו מספר רב של השאלות מספריות שונות. עבור כל השאלה יש לנהל את נתוני הזיהוי הכלליים של הסטודנט, תאריך השאלה, תאריך החזרה צפוי.

— עבור כל סופר יש לנהל את הנתונים הבאים:

* נתונים כלליים על הסופר (שם הסופר, קוד שפה, מין, שטחי התעניינות, מקום הולדתו, שנים בהם פעל).
* הספרים שהמחבר כתב ומופיעים במערכת.

— עבור כל ספריה יש לנהל את הנתונים הבאים:

* נתונים כלליים על הספריה (שם הספריה, כתובת).
* מלאי הספרים בספריה. בכל ספריה מספר רב של ספרים. עבור כל אחד יש לנהל את מספר העותקים התקני ומספר עותקים בפועל.
* השאלות שבוצעו בספריה. עבור כל השאלה יש לנהל את פרטי הספר המושאל ופרטי השואל.

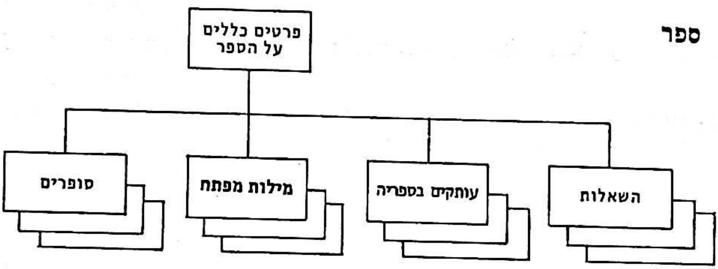
— עבור כל השאלה יש לנהל את הנתונים הבאים:

* נתונים על השואל (שם, מספר זהות, כתובת).
* הספרים הנמצאים ברשותו של השואל. עבור כל ספר יש לנהל את תאריך ההשאלה ותאריך החזרה צפוי.

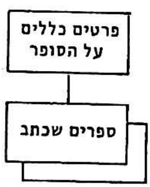
ב. בחירת מבנה נתונים

סביר להניח שעבור כל סוג נתונים אפשר להתאים מבנה לוגי של נתונים אשר מתאים לו ביותר. בתרשים 10-9 המבנים הלוגיים לאוכלוסיות הנתונים השונות.

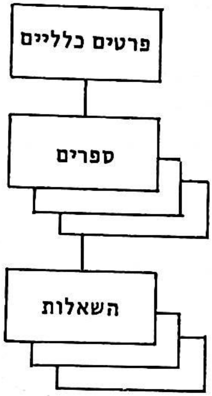
1. ספר



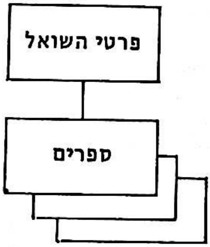
2. סופר



3. ספריה



4. השאלות



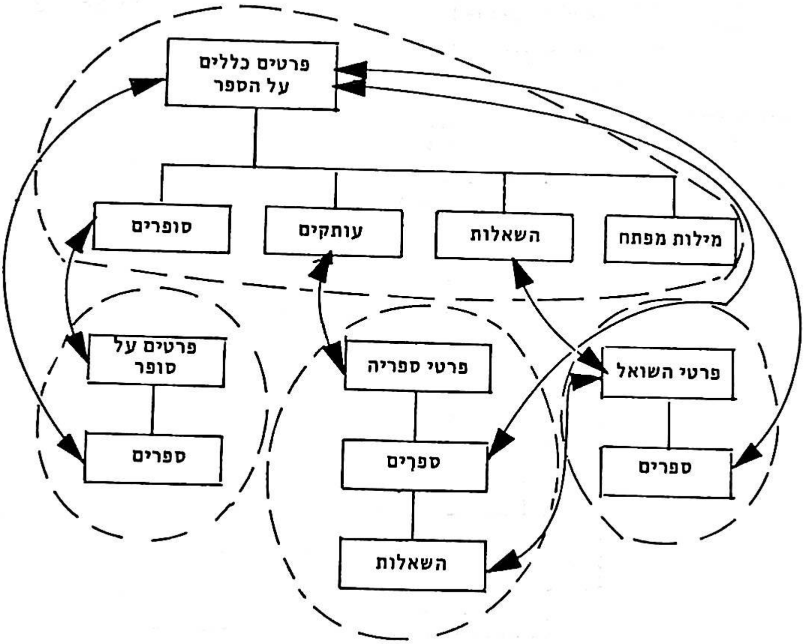
תרשים 10-9: מבני נתונים לקבצים שונים

כפי שניתן לראות קיימים קשרים רבים בין הרכיבים השונים. ניהול הנתונים במבנה כפי שהוצג, יגרום לכפילות נתונים גדולה ולבעיות חמורות בזמן עדכון. שיטת היישום תהיה להחזיק את הנתונים הכלליים במקום אחד בלבד ולהשתמש במפתחות שונים ליצירת הקשרים ביניהם. מאחר והנחנו פתרון בעזרת מערכת תוכנה לניהול קבצים לא הוצגו המצביעים השונים. קיימות מערכות לניהול בסיסי נתונים (כגון DL/I של יבמ) המאפשרות יישום פשוט של מבנים לוגיים אלה.

תרשים 10-10 מציג את מגוון הקשרים הקיימים בין הנתונים שבקבצים השונים.

ג. בחירת מודל פיסי

ניתן ליישם את המבנה הלוגי המוצג במספר מודלים פיסיים שונים ונציג לדוגמא אחד מהם. נחזור ונפרט את רשימת הקבצים במערכת, שיטות הארגון שלהם ותאור הרשומות המרכיבות אותם.



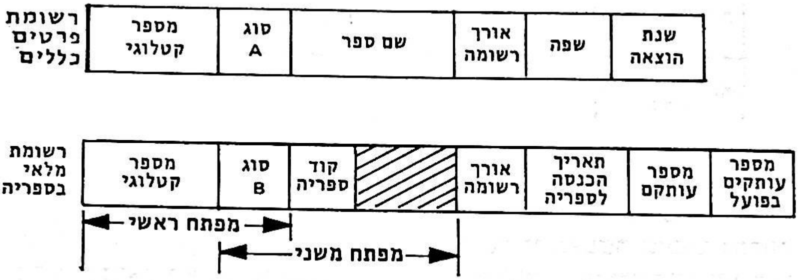
תרשים 10-10: קשרים בין נתונים בקבצים שונים

— קובץ סופרים יכיל את כל הפרטים על הסופרים המוכרים במערכת. הקובץ יאורגן כקובץ אינדקס סדרתי לייעול תהליך הפקת הדו"ח החודשי. לקובץ מפתח אחד — שם סופר.

שם סופר	קוד שפה	תרגום	שטחי התענינות	מקום הולדת	שנים בהם פעל

תרשים 10-11

— קובץ ספרים מכיל את הפרטים הכלליים של הספר ואת מספר העותקים בספריות השונות. הקובץ יאורגן בארגון ישיר עם פניה בעזרת חישוב. המפתח הראשי יהיה מספר קטלוגי מאחר וזה יהיה נתון הקשר עם רוב הקבצים האחרים ולכן הגישה בעזרתו תהיה המהירה ביותר. בנוסף יש לקובץ מפתח משני, שם ספר, המשמש בעיקר לשאילתות.



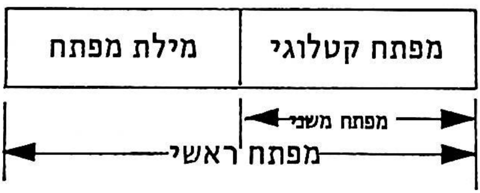
תרשים 10-12

— קובץ קשר. מאחר ולכל ספר יתכנו מספר סופרים וכל סופר יכול לכתוב מספר ספרים ניתן לנהל קשר זה בין סופרים לספרים בעזרת רשומת קשר מיוחדת. קובץ הקשר יאורגן במבנה אינדקס סדרתי כדי שניתן יהיה לקרוא אותו לפי מפתח חלקי (ראה תרשים זרימה של שאילתה).



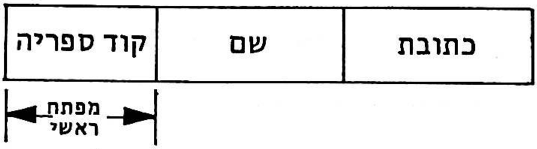
תרשים 10-13

— קובץ מילות מפתח יכיל את כל מילות המפתח המוכרות במערכת. הקובץ יהיה במבנה אינדקס סדרתי.



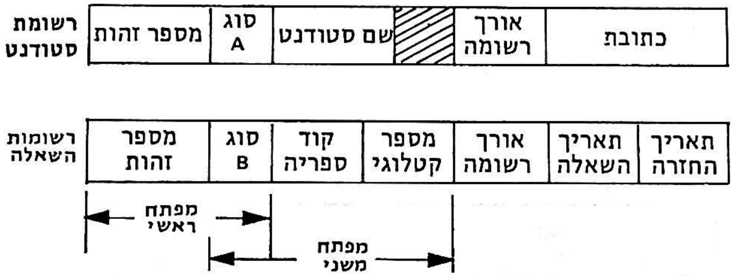
תרשים 10-14

— קובץ ספריות יכיל את הפרטים הכלליים של הספריה. מאחר ומספר הספריות קטן יחסית וניתן לבנות את סמל הספריה בצורה צפופה עולה, יאורגן קובץ זה בארגון ישיר עם יחוס ישיר.



תרשים 10-15

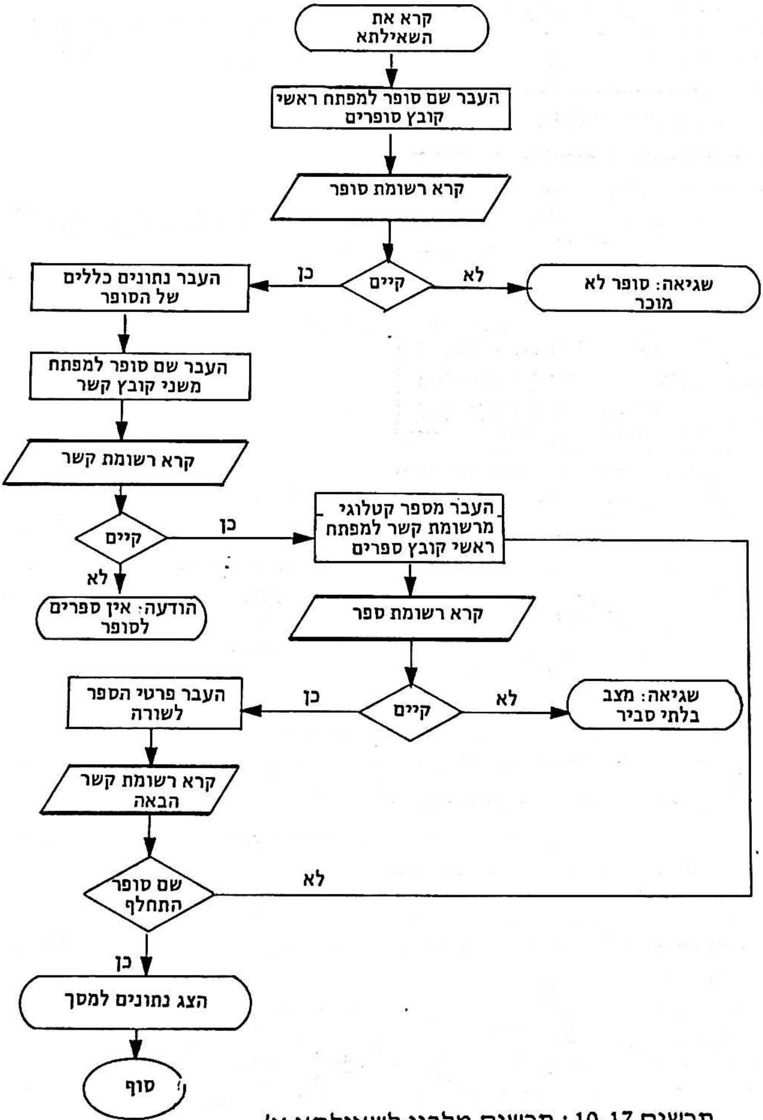
— קובץ השאלות יכיל את פרטי השואל ורשימת כל הספרים המושאלים לו. הקובץ יאורגן במבנה אינדקס סדרתי. שיטה זו תאפשר שימוש במפתח חלקי (לקראת רשימת כל הספרים של סטודנט למשל). הקובץ יכיל שני סוגי רשומות: רשומות סטודנט ורשומות השאלה.



תרשים 10-16

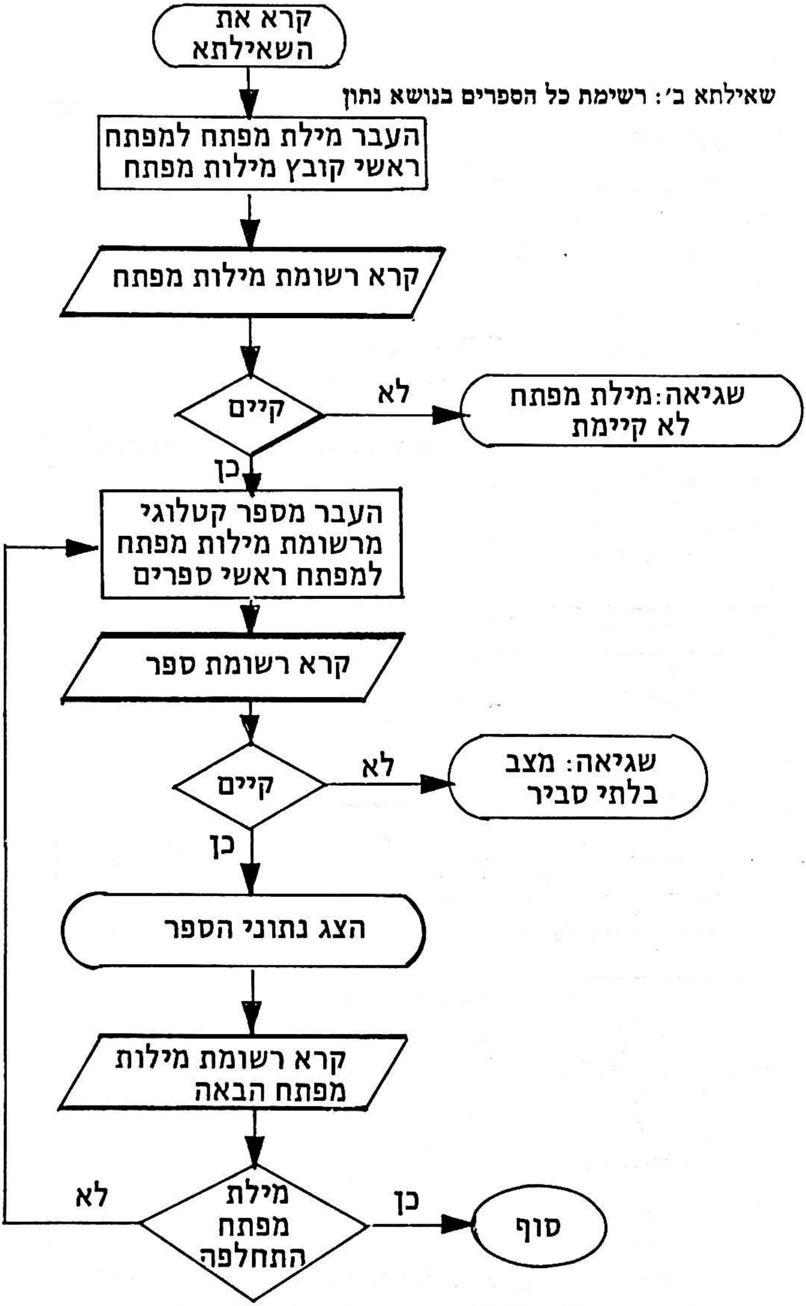
תרשימים 10-17 ו־10-18 מציגים תהליך לוגי של שאילתות. האחת מבקשת חתך לפי סופר והשניה — לפי נושא.

שאילתא א׳: רשימת ספרים שנכתבו על־ידי סופר.



תרשים 10-17: תרשים מלבני לשאילתא א׳.

שאילתא ב׳: רשימת כל הספרים בנושא נתון



תרשים 10-18: תרשים מלבני לשאילתא ב׳.

10.6 שאלות

1. מקובץ סטודנטים באוניברסיטה יש לשלוף את רשומות כל הסטודנטים הגברים הלומדים מתמטיקה וגילם מעל 21. הסבר כיצד תבצע שליפה זו עבור:
   - — קובץ סדרתי,
   - — קובץ מאורגן בגישה ישירה,
   - — קובץ אינדקס סדרתי,
   - — קובץ עם אינדקסים משניים.
2. ציין מהם המבנים המתאימים ביותר לדעתך עבור כל אחת מהמערכות הבאות:
   - — ניהול רשומות רפואיות של חולים במרפאה,
   - — מינויי טלפון,
   - — קובץ המתאר מבנה ארגוני,
   - — עץ מוצר,
   - — קובץ עובדים לתשלום שכר עבודה.
3. עליך לתכנן מערכת מידע לניהול רשומות רפואיות. אוכלוסית המבוטחים מונה כמיליון וחצי מבוטחים. רשומת מבוטח צריכה להכיל בין 200 ל־400 תווים (שם, גיל, כתובת ועוד). בממוצע יש לכל מבוטח את הנתונים הבאים:
   - — חמישה ביקורים בבית המרקחת. בכל ביקור הוא מקבל 2.8 תרופות. כל רשומה תרופה מכילה 30 תווים.
   - — 0.8 אשפוזים. רשומת אשפוז באורך 1000 תווים.
   - — 2.4 ביקורים אצל רופא. אורך רשומה 70 תווים.

   המערכת משתמשת ביחידת אחסון המאפשרת בניית גושים בגודל 7000 תווים כל אחד. זמן חיפוש ממוצע 25 מילישניות, השהיה סיבובית 10 מילישניות, מהירות העברת נתונים 0.8 מיליון בתים לשניה. בכל מסילה 5 גושים. 50 מסילות לגליל.

   א. תכנן את הקבצים ושיטות הארגון הדרושים למערכת.

   ב. חשב את גודל הרשומות בכל הקבצים ואת שטח האחסון הכולל הדרוש למערכת.

   ג. חשב את הזמן הדרוש לשליפת רשומת ביקור במרפאה לפי שם מבוטח ותאריך ביקור.

   ד. חשב את הזמן הדרוש לסריקת הקובץ למציאת מספר המבוטחים המשתמשים בתרופה מסוימת.

פרק אחד־עשר
מערכות לניהול קבצים

11.1 מבוא
11.2 שיטות עבודה במערכות לניהול קבצים
11.3 ניהול קבצים ביחידות אחסון בגישה ישירה
11.4 אבטחה ופרטיות בניהול קבצים
11.5 סיכום
11.6 שאלות

בפרק זה נציג את המודל הכללי של מערכות לניהול קבצים ונסביר את שיטות העבודה של מערכות אלו.

11.1 מבוא

המערכת לניהול קבצים (FMS — File Management System) היא התוכנה המאפשרת את הגדרת הקבצים והטיפול בהם. זוהי אחת המערכות החשובות אשר בלעדיה נושא ארגון הנתונים היה הופך למורכב ביותר. לרוב מקבל המשתמש את התוכנה מיצרן המחשב או מבית תוכנה, ולעיתים רחוקות הוא כותב תוכנה זו בעצמו כפי שמקובל במחשבים זעירים. המגוון הרחב של מבנים פיסיים וריבוי סוגי יחידות האחסון הופך את משימת ניהול הנתונים למורכבת ביותר. הבעיות העיקריות הן:

א. מגוון רחב של מבנים פיסיים: קיים מספר גדול יחסית של מבנים פיסיים לייצוג נתונים שיש להסב ולהתאים למבנה נתונים לוגי בכדי לבצע את העיבוד. המבנים הפיסיים תלויים ביחידות האחסון.

ב. נפח אחסון ומהירויות העברת נתונים: יחידות האחסון השונות הן בעלות נפחים ומהירויות העברת נתונים שונים. יש להתחשב בנתונים אלו בעת תכנון הקבצים ולהציב אותם ביחידות האחסון בהתאם לנפח האחסון הדרוש ובהתאם למידת הפעילות.

ג. יחידות העברת נתונים: העברת הנתונים בין המחשב ויחידות האחסון נעשית ביחידות שונות. היא יכולה להתבצע בגושים, בבתים או במילים בהתאם לסוג המחשב וסוג יחידת האחסון הקשורה אליו. במחשבים גדולים העברת הנתונים הינה בגושים.

ד. צורת ייצוג הנתונים: לעיתים נתונים מיוצגים בצורה שונה ביחידות אחסון שונות ולכן יש צורך בביצוע תרגום לקודים הנדרשים בעת העיבוד.

ה. מגוון רחב של פעולות קלט/פלט: ליחידות האחסון מגוון רחב של פעולות קלט/פלט. רוב הפעולות מותאמות לסוג יחידת האחסון כגון גלגול סרט מגנטי קדימה, הזזת זרוע דיסק וכדומה. בתוך קבוצת יחידות האחסון לגישה ישירה יש מגוון רחב של פעולות בהתאם לסוג המחשב וסוג היחידה הקשורה אליו.

ו. מגוון מצבים חריגים: על המערכות לטפל במגוון של מצבים חריגים בזמן ביצוע פעולות הקלט/פלט, כגון: סוף סרט מגנטי, שגיאות בהעברת נתונים בין יחידת האחסון וזכרון המחשב (Parity Error למשל), חיפוש עם מפתח שאינו קיים באינדקס ועוד.

על מנת להקל את משימת ניהול הנתונים עבור המשתמש מספקים יצרני המחשבים תוכנה סטנדרטית לניהול הנתונים אשר כוללת אפשרויות התאמה רבות לצרכיו של כל משתמש. מטרות תוכנה זו הם:

א. לאפשר למשתמש מגוון מסוים של מבנים לוגיים ופיסיים המאפשרים עיבוד נוח של הנתונים תוך שימוש יעיל ביחידות האחסון.

ב. תמיכה במגוון רחב של יחידות אחסון בעלות ביצועים ועלויות שונות. מערכות ניהול נתונים משחררות לחלוטין את המשתמש מפנייה לקובץ פיסי. התוכניות אינן קשורות ליחידת אחסון מסוימת אלא פונות **ליחידת אחסון בפועל** (Virtual Device). הקשר בין יחידת האחסון הפיסית ליחידה בפועל מתבצע באמצעות מערכת ההפעלה על סמך נתונים הנמסרים לה על ידי המשתמש בעזרת **כרטיסי פיקוח** (JCL — Job Control Language).

ג. הגדלת יעילות פעולות קלט/פלט. מערכות ניהול הנתונים מטפלות באחסון, בקריאה ובכתיבה של גושים לקבלת יעילות עיבוד מירבית.

ד. תרגום קודים ושחרור המשתמש מהתייחסות לייצוג הפנימי של הנתונים.

בפרק זה נציג מודל כללי לשיטת העבודה של מערכת ניהול נתונים וכמה מהשיטות השונות לניהול הקבצים ביחידות האחסון.

11.2 שיטות עבודה במערכות לניהול קבצים

קיימות שתי יחידות תוכנה המטפלות בניהול הנתונים — **תוכנית המשתמש** הקובעת את המבנה הלוגי של הנתונים ואת דרישות העיבוד, **והתוכנה לניהול קבצים** המטפלת בתרגום המבנה הלוגי למבנה פיסי, באחסון, בפעולות קלט/פלט ובחריגים.

התקשורת בין שתי יחידות אלה מתבצעת בעזרת **טבלת תיאור קבצים** אשר נכנה אותה FDT (File Description Table). במערכות יבמ היא נקראת **טבלת בקרת נתונים** — DCB (Data Control Block) ובמערכות סי.די.סי. היא נקראת **טבלת מידע קבצים** — FIT (File Information Table). טבלה זו שייכת לתוכנית ונמצאת בזיכרון בזמן העיבוד.



תרשים 11-1: טבלת תיאור קבצים (FDT)

טבלת תיאור קבצים

טבלת תיאור קבצים מכילה אוסף של פרמטרים ושטחים משותפים לשתי יחידות התוכנה והיא מקשרת ביניהן. יש בה את כל הפרמטרים הדרושים להגדרת מבנה הקובץ ולפחות הפרטים הבאים בהתאם למבנה הקובץ:

— שיטת ארגון הקובץ,
— גודל הגוש וסוג הגוש (אורך קבוע, משתנה, לא מוגדר),
— גודל הרשומה, סוג הרשומה, אורך מירבי ומזערי, מספר רשומות בגוש,
— מיקום המפתח (עבור שיטות ארגון לא סדרתית),
— תיאור תווית (Label) הקובץ,
— יחידות אחסון שהקובץ רשום בהן.

תוכניתן הכותב בשפת סף (Assembler) חייב לדאוג לבניית הטבלה כאשר הוא נעזר בהוראות מקרו מיוחדות. תוכניתן הכותב בשפה עילית אינו מתיחס לטבלה זו באופן ישיר, אלא כותב פרמטרים כלליים שמהם נבנית הטבלה הדרושה.

להגדרת המבנה הלוגי של הקובץ בשפה עילית משתמשים **בשפת פרמטרים להגדרת נתונים** (DDL — Data Definition Language). המהדר (Compiler) מכיל בתוכו שגרות המתרגמות פרמטרים אלו לטבלת תיאור הקבצים (FDT). שגרות אלו קרויות גם **מישק טבלת תיאור קבצים** (FDT Interface).

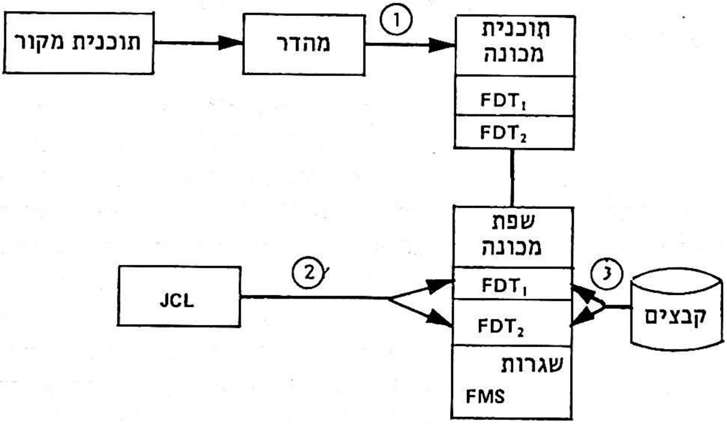


תרשים 11-2: בנית FDT בעזרת מהדר

הפרטים המלאים בטבלת תיאור קבצים נרשמים בשלשה מועדים שונים:
— בעת הידור התוכנית,
— בעת טעינת התוכנית לזיכרון,
— בעת ביצוע התוכנית ("ריצה" — Run).

המהדר בונה את טבלת התיאור עבור כל קובץ המוגדר בתוכנית. קיימים פרטים שאינם ידועים בזמן ההידור, כמו לדוגמא באיזו יחידת אחסון ימצא הקובץ, בסרט או בדיסק ובאיזה דיסק. הגדרה מפורשת היתה מחייבת הידור מחדש בכל פעם שיחידת האחסון משתנה (במידה שניתן לשנותה). נתון זה נמסר רק בזמן קריאת כרטיסי הפיקוח המפעילים את התוכנית, ולכן הוא נרשם בטבלת תיאור הקובץ רק בזמן טעינת התוכנית לזיכרון. בנוסף קיים מידע הנמסר רק בזמן הריצה, לדוגמא התווית של הקובץ ידועה רק לאחר שפתחנו אותו.

לאחר פתיחת הקובץ (OPEN) מכילה טבלת תיאור הקובץ את כל הפרמטרים הדרושים לעיבוד הקובץ בעזרת מערכת ניהול הקבצים. תרשים 3-11 מדגים את התהליך של מילוי הנתונים בטבלה.

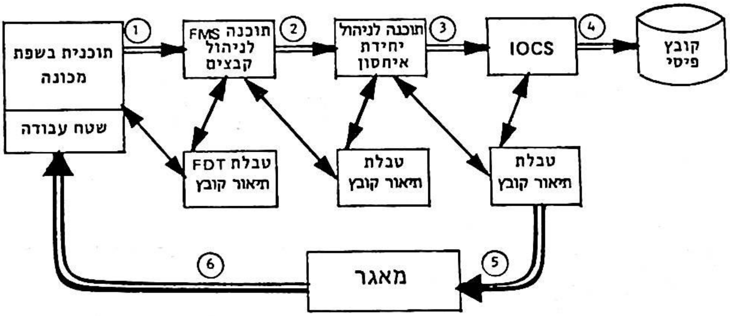


תרשים 11-3: השלמת נתונים בטבלת תיאור הקובץ בזמן הידור (1) בזמן טעינה (2) ולאחר פתיחת הקובץ (3)

מודל כללי לניהול קבצים

מערכת התוכנה לניהול קבצים (FMS — File Management System) אינה תוכנה סטנדרטית כדוגמת שפת תכנות קובול או פורטרן למשל. בעוד ששפות אלו דומות מאד מכיון שחלק מהאיפיון שלהן הינו סטנדרטי, הדמיון שישנו בין מערכות התוכנה לניהול קבצים הינו במישור הפונקציונלי בלבד. יש שוני רב מאד בין מערכות התוכנה של יצרנים שונים ואפילו בין מערכות תוכנה של אותו יצרן אשר פועלות במחשבים שונים. עם זאת ניתן ללמוד ולהכיר את מודל הפעולה של מערכת ניהול קבצים ואחר־כך ללמוד ולהכיר את התכונות המיוחדות לכל מערכת תוכנה שבה חמשתמשים.

תיאור מפורט של המודל ניתן בתרשים 4-11 ובהסבר הנלווה.



תרשים 4-11: תזרים העבודה במערכת תוכנה לניהול קבצים

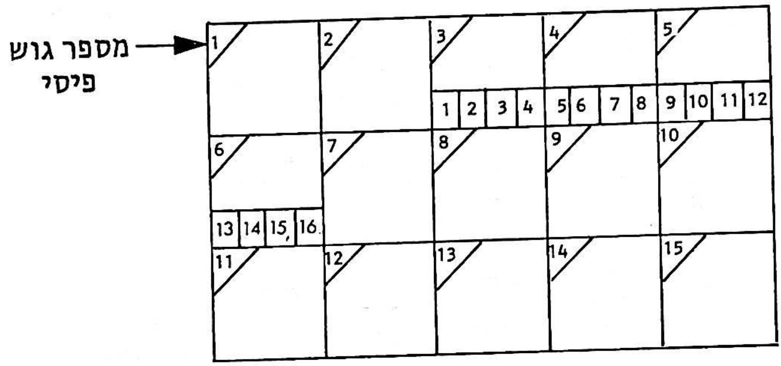
1. תוכנית המשתמש מבצעת פקודת ניהול קובץ (DML — Data Management Language) ובשלב זה עובר הפיקוח ל-FMS. תוכנה זו מפענחת את הבקשה. לאחר הפענוח היא משתמשת בפרמטרים הנמצאים ב-FDT כדי להשלים אותה בנתונים על הקובץ הפיסי. אם הבקשה היתה קריאת רשומה, מבוצע חישוב של הגוש הלוגי בתוכו צריכה להמצא הרשומה המבוקשת. לפעמים כדי לבצע תרגום זה על FMS לבצע חיפושים באינדקסים. מספר הגוש הלוגי הוא המספר היחסי מתחילת הקובץ.
2. מספר הגוש הלוגי נמסר לתוכנה לניהול יחידות האחסון. תוכנה זו מבצעת את המיפוי בין המספר היחסי של הגוש לכתובת המוחלטת ביחידת האחסון. המיפוי מתבצע באמצעות טבלאות מיוחדות אשר תוסברנה בהמשך.
3. הכתובת וסוג פעולת קלט/פלט נמסרים לשגרות בקרה של מערכת ההפעלה המטפלות בכל הגישות הפיסיות ליחידות האחסון (IOCS — Input/Output Control System).
4. מתבצעת גישה פיסית לנתונים בהתאם לכתובת המוחלטת שחושבה.
5. הגוש המבוקש מועבר אל המאגר המתאים בזכרון ונמסרת הודעה ל-FMS שהגוש נמצא במאגר.

6. על פי פרמטרים הנמצאים בטבלת התיאור (FDT) שולפת התוכנה לניהול קבצים (FMS) את הרשומה המבוקשת מתוך המאגר ומעבירה אותה לשטח העבודה של תוכנית המשתמש.

7. תוכנית המשתמש מקבלת שוב את הפיקוח וממשיכה לעבד את הרשומה.

נבהיר את שיטת העבודה בעזרת דוגמא המוצגת בתרשים 11-5: נניח שקובץ מכיל 15 רשומות, אורך כל רשומה 250 בתים, 4 רשומות לגוש, הקובץ תופס 4 גושים רצופים. מבחינה לוגית הקובץ מכיל 4 גושים — 1, 2, 3, 4, אך מבחינה פיסית גושים אלה הם 3, 4, 5, 6. נניח שתוכנית המשתמש מבקשת לקרוא את הרשומה השישית בקובץ באמצעות ההוראה הבאה:

READ file-name RECORD (6)



תרשים 11-5: פריסת הקובץ לדוגמא ביחידת האחסון

נעקוב אחר תהליך ביצוע ההוראה:

1. חישוב הגוש הלוגי המכיל את הרשומה השישית. גוש מכיל 4 רשומות, ולכן הרשומה השישית נמצאת בבלוק הלוגי השני.

2. התוכנה לניהול יחידת האחסון מקבלת את השם הסמלי של הקובץ ובקשה להביא לזכרון את הגוש הלוגי השני.

תוכנה זו מבצעת את התרגום לגוש הפיסי בעזרת טבלה מיוחדת VTOC (Volume Table Of Contents) המציינת עבור כל קובץ את המקום שבו הוא מאוחסן מבחינה פיסית ומה אורכו.

הגוש הפיסי יחושב כ(גוש לוגי + נקודת התחלה — 1). מתוך חישוב זה נקבל שהגוש הפיסי המתאים הוא גוש מספר 4:

(2 + 3 — 1 = 4)

3. הכתובת הפיסית של גוש מספר 4 נמסרת ל-IOCS.

4. מתבצעת קריאה של גוש 4 מהדיסק של המאגר.

5. נמסרת הודעה למערכת התוכנה לניהול קבצים (FMS) שהגוש המבוקש נמצא בזכרון.

6. התוכנה לניהול קבצים מחשבת את המקום הסידורי של הרשומה השישית בתוך הגוש. נשתמש בנוסחה הבאה:

מקום סידורי בגוש = מספר רשומה — (מספר גוש לוגי — 1) × מספר רשומות בגוש

$$2 = 6 - (2 - 1) \times 4$$

נציב את הנתונים ונקבל שהמקום הסידורי הוא 2. כלומר, יש לשלוף את הרשומה הלוגית השניה מתוך הגוש. יש לחשב עתה את הכתובת היחסית בתוך הגוש של הרשומה השנייה:

כתובת התחלתית = (מספר סדורי של הרשומה — 1) × אורך הרשומה + 1

$$251 = (2 - 1) \times 250 + 1$$

אם נציב נקבל 251. מערכת FMS תעביר לשטח העבודה של התוכנית 250 בתים החל מהבית 251 של הגוש.

7. תוכנית המשתמש מקבלת פיקוח וממשיכה לפעול.

בדוגמא זו, עבור כל פקודת קריאה תתבצע גישה לדיסק לקריאת הגוש המבוקש. מאחר ובגוש יש 4 רשומות ניתן לשפר בקלות את האלגוריתם. מערכת ניהול הקבצים תנהל רישום של מספר הגוש הנמצא המאגר, ולפני הגישה לקבלת גוש חדש תבדוק תחילה אם הגוש כבר נמצא בזיכרון. אם הוא נמצא, יבוצע מיד דילוג לשלב 6 באלגוריתם שהוצג. אם הגוש אינו נמצא בזכרון תבוצע תחילה פעולת קריאה פיסית. בצורה זו לקריאת כל הקובץ דרושות 4 גישות פיסיות לדיסק לעומת 15 גישות בשיטה הקודמת.

פעולות עיבוד בקבצים

המשתמש יכול לבצע פעולות עיבוד שונות כמו פתיחה וסגירה של קובץ, קריאה וכתיבה, איתור רשומות ועוד. פעולות אלו תלויות במבנה הקובץ ובמערכת לניהול קבצים שבה משתמשים. הפקודות האופייניות למערכת לניהול קבצים מוצגות כהוראתן בקובול:

— OPEN: פקודה זו מבצעת את **פתיחת הקובץ** לפני עיבודו. בפקודה זו מודיעים למערכת אילו פעולות התוכנית תבצע בקובץ (קריאה בלבד, כתיבה בלבד, קריאה וכתיבה). פקודה זו מעתיקה מתוך הקובץ לזיכרון את הטבלאות הדרושות ודואגת להקצאת מאגרים בזיכרון.

— CLOSE: פקודה זו מבצעת את **סגירת הקובץ** בגמר עיבודו. אם המאגרים אינם ריקים, הם נכתבים ליחידת האחסון, הטבלאות השונות מתעדכנות ושטח המאגרים משתחרר.

— GET: פקודה זו מאפשרת **קריאת רשומה לפי מפתח** ועל־כן אינה אפשרית בארגון סדרתי. הפקודה מכילה את מפתח הרשומה המבוקשת. בהתאם לשיטת ארגון הקובץ מבצעת המערכת את הפעולות הדרושות לאיתור הרשומה. אם הרשומה המבוקשת לא נמצאת בקובץ תועבר הודעה מתאימה לתיכנית המשתמש.

— GET-NEXT: פקודה זו מאפשרת **קריאת הרשומה הבאה** בהתאם למיקום הנוכחי בקובץ. המיקום מתבצע בעזרת פקודת GET או START. אם המיקום הוא בתוך אינדקס משני תתקבל הרשומה הבאה בהתאם לאינדקס המשני. ניתן להשתמש בפקודה זו לסריקת הקובץ מתחילתו ועד סופו.

— START: פקודה זו מאפשרת **מיקום התוכנית** בקובץ אינדקס סדרתי או באינדקס המשני. בדרך כלל ניתן לנסח במסגרת הפקודה תנאי (Condition) לקביעת המיקום, לדוגמא התמקם על הרשומה שהמפתח שלה אינו קטן מ־100. פקודה זו אינה מעבירה את הרשומה מהמאגר אל שטח העבודה ולכן יש לבצע **מיד** פקודת GET או GET-NEXT.

— WRITE: פקודה זו מאפשרת **כתיבת רשומות חדשות** לקובץ. המערכת תבדוק לפני הכתיבה אם קיימת רשומה עם מפתח כזה ולא תבצע את הכתיבה. בארגון סדרתי לא נעשית בדיקה כי הכתיבה רציפה.

— REWRITE: פקודה זו מאפשרת **עדכון רשומה קיימת על־ידי כתיבתה במקום הרשומה הקודמת**. את הרשומה יש לקרוא תחילה בעזרת פקודת GET או GET-NEXT.

— DELETE: פקודה זו מאפשרת **ביטול רשומה** קיימת למעט אירגון סדרתי. המערכת תמסור הודעה לתוכנית המשתמש אם הרשומה לא נמצאת בקובץ.

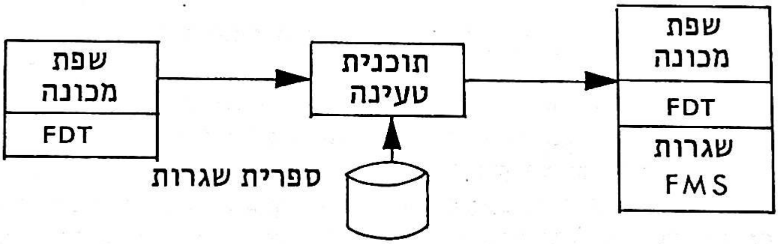
— SEEK: פקודה זו מאפשרת התחלת תהליך **חיפוש רשומה לפי מפתח** במקביל לעבודת התוכנית. הרשומה המבוקשת מועברת אל המאגר במקביל להתקדמות התוכנית. היא תועבר אל שטח העבודה של התוכנית רק עם ביצוע פקודת GET. השימוש בפקודה מאפשר עיבוד חופף.

טעינת שגרות FMS לזכרון

כפי שניתן לשער, שטח הזיכרון שנדרש לשגרות FMS עלול להיות גדול מאד בגלל היותן כלליות מאד ומתאימות למגוון מבנים פיסיים ולוגיים ויחידות אחסון שונות. בפועל אין צורך לטעון את כל שגרות FMS עם כל תוכנית אלא לטעון רק את השגרות הדרושות לעיבוד הקבצים שבהן משתמשת התוכנית. קיימות שתי שיטות לטעינת שגרות מערכת התוכנה לניהול קבצים אשר נתאר בהמשך.

א. טעינה סטטית

בזמן טעינת תכנית המשתמש לזיכרון מופעלת תוכנית טעינה מיוחדת (Loader) של מערכת ההפעלה. תוכנית זו סורקת טבלה שהוכנה על-ידי המהדר ובה רשומים המבנים המטופלים בתוכנית ודרישות הגישה (קריאה בלבד, קריאה וכתיבה). תוכנית הטעינה שולפת את השגרות הדרושות מתוך ספריית השגרות ומצרפת אותן אל תוכנית המשתמש. בשיטה זו לכל תוכנית, העתק נפרד עם כל שגרות FMS הדרושות לה בזמן הביצוע, כפי שמודגם בתרשים 11-6.

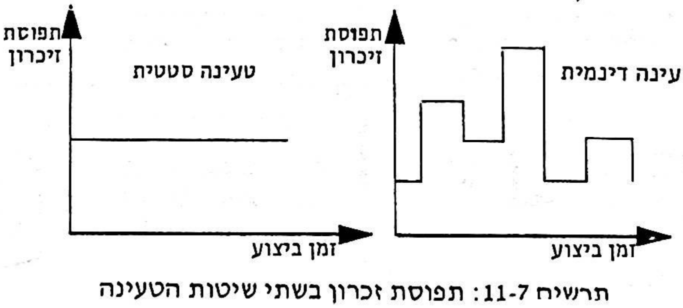


תרשים 11-6: טעינה סטטית

ב. טעינה דינמית

שגרות FMS אינן נדרשות במשך כל זמן ביצוע התוכנית ולכן ניתן להביאן לזיכרון בעת הצורך ועם גמר הטיפול לשחרר מיד את שטח הזיכרון. אפשר להיות סלקטיביים ולהחזיק בזיכרון את השגרות שהשימוש בהן רב, כמו קריאה וכתיבה למשל, ולשחרר את השגרות המטפלות בפתיחת קובץ אשר אינן דרושות לאחר פתיחתו.

תמונת הזיכרון תהיה שונה בשתי שיטות הטעינה כפי שמוצג בתרשים 11-7.



תרשים 11-7: תפוסת זכרון בשתי שיטות הטעינה

11.3 ניהול קבצים ביחידות אחסון לגישה ישירה

האפשרויות השונות למבני נתונים ביחידות אחסון לגישה ישירה מציגות אפשרויות שונות לניהול הקבצים הפיסיים ביחידות אלו. מבחינת מערכת ההפעלה אשר מנהלת את יחידות האחסון, הקובץ הינו אוסף של גושים המכילים נתונים. הארגון הלוגי שלהם ותוכנם אינו בשליטת מערכת ההפעלה. הקובץ יכול להכיל נתונים, תוכניות מחשב, נוהלי פיקוח (JCL), ועוד. הקובץ הוא יחידה לוגית שיש לנהל אותה ללא קשר לתוכנה.

התפקידים העיקריים של מערכת ניהול יחידות האחסון בגישה ישירה הם:

א. לאפשר בניית קבצים חדשים בעלי גודל כלשהו וביטולם בעת הצורך.

ב. לתת אפשרות גישה לקבצים לשם קריאה או כתיבה. לעיתים יש לאפשר למספר משתמשים לגשת בו זמנית לאותו קובץ.

ג. מנוהלים הגושים שאינם מופיעים בצורה רציפה ביחידת האחסון. המערכת צריכה לשחרר את המשתמש מידיעת המיקום הפיסי של הקובץ.

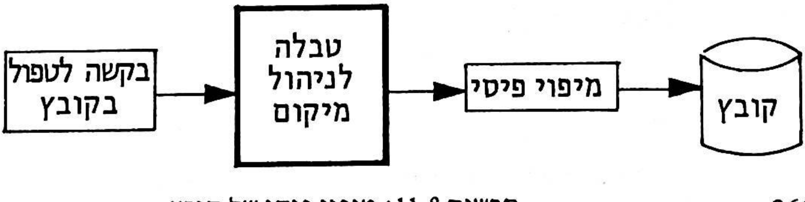
ד. לאפשר פניה לקובץ תוך שימוש בשם סימלי.

ה. להגן על הקבצים בזמן תקלה.

היחידה הבסיסית מבחינת המערכת לניהול יחידות אחסון הוא הגוש. ברוב המערכות גודל הגוש נקבע מראש על-ידי היצרן ואינו ניתן לשינוי על-ידי המשתמש. מערכות יבמ הן דוגמא למערכות בהן יש למשתמש שליטה על גודל הגוש. השיקולים העיקריים של היצרנים בקביעת גודל הגוש הם שיקולי מבנה הדיסקים ויעילות פעולות קלט/פלט תוך ניצול יעיל של שטחי האחסון. גדלים מקובלים הם 128, 256, 512 תווים לגוש. כאשר המשתמש מגדיר בתוכנית גושים באורך שונה (גושים לוגיים) הם נרשמים בגוש פיסי אחד או במספר גושים פיסיים. אם גודל הגוש אינו כפולה שלימה של הגודל הסטנדרטי, מבצעת המערכת באופן אוטומטי השלמה של השטח הנותר בסימן מוסכם. בשיטה זו יש אלמנט מסוים של בזבוז שטח אחסון.

טבלאות ניהול קבצים

הקישור בין השם הסימלי של הקובץ לבין מיקומו הפיסי ביחידת האחסון מתבצע בעזרת טבלת הדרכה מיוחדת (File Directory) המכילה כניסה אחת או יותר עבור כל קובץ ביחידת האחסון. זוהי הטבלה לניהול מיקום הקובץ (במערכות יבמ קרויה טבלה זו VTOC — Volume Table Of Contents).



תרשים 11-8: מיפוי פיסי של קובץ

טבלה זו מכילה את הנתונים הבאים:

— השם הסימלי של הקובץ,
— המיקום הפיסי שלו ביחידת האחסון,
— סוג הנתונים בקובץ (בינרי, דצימלי),
— נתונים לאבטחת הגישות לקובץ, כגון סיסמאות,
— נתונים מנהליים וסטטיסטיים כגון: תאריך יצירה, תאריך גישה אחרון, גודל ועוד.

המשתמש מודיע למערכת ההפעלה על צורת עיבוד הקובץ באמצעות כרטיסי פיקוד (JCL). לתוכן ולמבנה הלוגי אין כל חשיבות מבחינת המערכת. מערכת ההפעלה כוללת שלש פקודות בסיסיות לניהול קבצים (פקודות אלה מיושמות בצורה שונה ב-JCL של כל מחשב):

CREATE — יצירת קובץ חדש,
ATTACH — שימוש בקובץ קיים,
DELETE — ביטול קובץ קיים.

שיטת הקישור בין השם הסימלי למיקומו הפיסי תלויה במידת התחכום של המערכת. נציג כאן מספר אפשרויות.

א. אחריות המיפוי מוטלת על המשתמש:

המשתמש צריך לקבוע את מיקום הקובץ ואת גודלו מראש. פקודה אופיינית למערכות מסוג זה:

CREATE, filename, LENGTH = 3750, CYLINDER = 3,
TRACK = 10, UNIT = ...

מבחינת המשתמש שיטה זו אינה נוחה כיוון שהמיקום הפיסי של הקובץ ביחידת האחסון נקבע בצורה ישירה על ידיו. אם יש מספר משתמשים באותה יחידת אחסון יש צורך בתיאום כדי שהאחד לא יהרוס את קבציו של האחר.

בדוגמא שלמעלה תקצה המערכת 3750 בתים רצופים החל מהכתובת הנתונה המוגדרת ב-CYLINDER, TRACK. אם המקום כבר הוקצה לקובץ אחר לא תתקבל על כך כל הודעה. אין אפשרות להגדיל את הקובץ מעבר למה שהוקצה לו בתחילה.

ב. מיפוי סטטי באחריות המערכת

קיימות מספר שיטות מיפוי המנוהלות באופן אוטומטי על־ידי המערכת. אחת האפשרויות היא לנהל טבלה שבה מופיעים כל השטחים הרציפים הפנויים ביחידת האחסון. קביעת מיקום פ סי לקובץ חדש מתבצעת תוך סריקת הטבלה ואיתור השטח הדרוש עבורו. גם בשיטה זו אין הקובץ יכול לעבור את הגודל שהוקצה לו בראשונה. פקודה אופיינית למערכות מסוג זה היא:

CREATE, file-name, LENGTH = 3750

כאשר אין שטח רציף פנוי אי אפשר להקצות את הקובץ. בכמה מערכות מתבצעת הזזה אוטומטית של קבצים לקבלת השטח הרציף הדרוש וזהו תהליך שאינו יעיל.

ג. מיפוי דינמי באחריות המערכת

המערכת מבצעת הקצאת שטחים דינמית. זוהי שיטת העבודה של רוב המערכות המתקדמות.

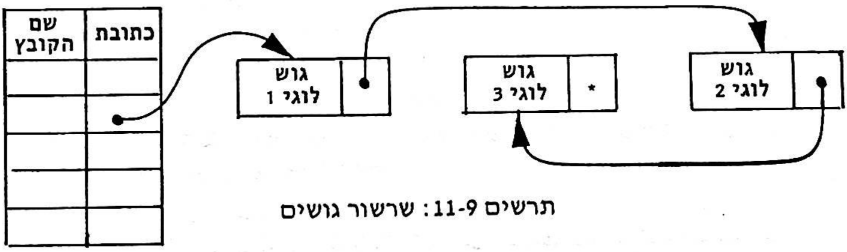
במערכות אלה אין צורך בהגדרת גודל הקובץ. תפקיד המערכת לאפשר גידול דינמי בהתאם לצורך. פקודה אופיינית למערכות מסוג זה:

CREATE, file-name

קיימות מספר שיטות לביצוע המיפוי הדינמי:

1. שרשור גושים

בכל גוש קיים מצביע אל הגוש הבא. בטבלת המיפוי מציינים את הכתובת של הגוש הראשון בקובץ, כפי שמודגם בתרשים 11-9.



תרשים 11-9: שרשור גושים

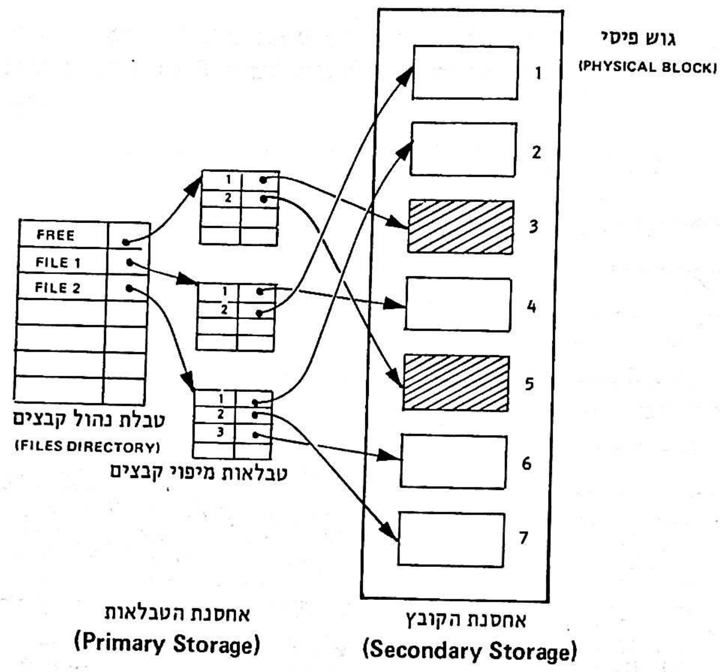
הגדלת הקובץ מתבצעת על־ידי שרשור גוש ריק חדש לשרשרת הקיימת ועל־כן הוא יכול להיות במקום פיסי כלשהו ביחידת האחסון. חסרון השיטה הוא במספר הגישות הרב שיש לבצע לאתור גוש כלשהו או כדי להגיע לסוף הקובץ. ביטול הקובץ מחייב קריאת כל הגושים ושרשורם שוב לשרשרת הגושים הפנויים. שיפור מסויים יושג אם בטבלת המיפוי נצביע ישירות לגוש האחרון בנוסף לגוש הראשון. שיטה זו נוחה לטיפול בקבצים סדרתיים, בהם העיבוד מתבצע על־ידי סריקת כל הקובץ.

2. מיפוי קובץ

בטבלת מיפוי הקובץ (File Map) מחזיקים מצביע לטבלה אחרת המבצעת מיפוי בין מספר הגוש הלוגי למספר הגוש הפיסי. גם בשיטה זו יכולים הגושים הפיסיים להיות מפוזרים בכל יחידת האחסון. בשיטה זו אין משרשרים את הגושים השייכים לאותו הקובץ. לניהול הגושים הפנויים משתמשים במפה נפרדת המכילה את כל הגושים הבלתי מנוצלים ביחידת האחסון. הוספת גוש חדש לקובץ מתבצעת על־ידי ביטולו ממפת הגושים הפנויים והוספתו למפת הקובץ.

תרשים 11-10 מציג את שיטת המיפוי: קובץ FILE 2 המכיל שלשה גושים לוגיים אשר נמצאים בגושים פיסיים 2, 7, 6.

השיטות המשתמשות במפות הגושים מאפשרות גידול דינמי של הקובץ. ביטול קובץ מתבצע בצורה נוחה מאחר ודרושה העברת מספרי הגושים הפיסיים ממפת הקובץ למפת הגושים הפנויים (FREE) בלבד. הוספת גוש



תרשים 11-10: שימוש במפת קובץ

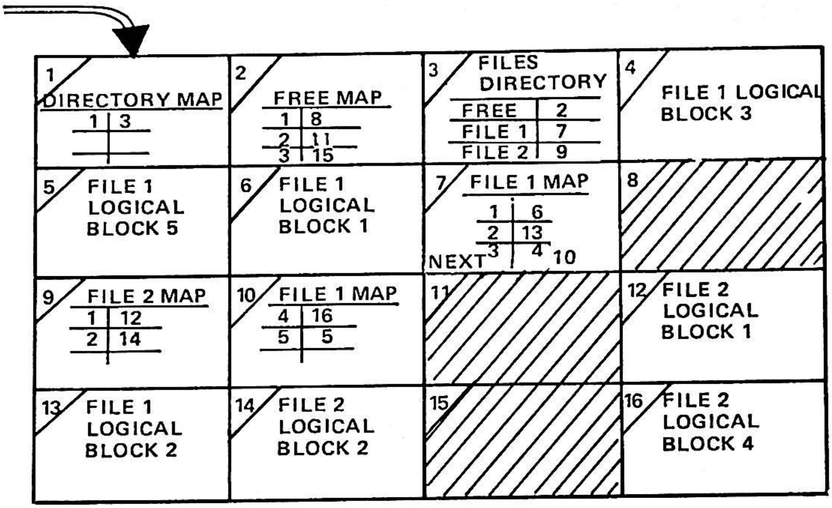
פנוי לקובץ יכולה להתבצע במספר אופנים. קיימות מערכות המשתמשות בגוש הראשון המופיע במפת הגושים הפנויים. לעומתן קיימות מערכות מתוחכמות המנסות להקצות את הגוש שיביא למינימום את מספר הזזות הזרוע בזמן קריאת הקובץ. פיזור אקראי של הגושים השייכים לקובץ ביחידת האחסון יגרום למספר רב מאד של הזזות זרוע בזמן קריאתו. באותם מקרים בהם שיטות ארגון הקובץ דורש הקצאת מראש של מספר גושים, כמו ארגון ישיר, תשתדל המערכת להקצות את כולם בצורה רציפה עד כמה שניתן.

3. מיפוי כפול

המפה של הקובץ וטבלת המיפוי הראשית יכולות להיות בעצמן גדולות ולכן שמירתן בזכרון מהווה בזבוז. ניתן להתייחס לטבלאות אלה כאל קבצים רגילים ולבנות טבלת מיפוי גם לטבלאות אלה. אם מפת הקובץ גולשת בעצמה מעבר לגוש אחד ניתן לשרשר את הגושים שלה. תרשים 11-11 מציג שיטת מיפוי זו.

בדוגמא יש ל-Directory גם כן מפה הנמצאת בגוש הפיסי הראשון. הטבלה עצמה נמצאת בגוש פיסי 3 ומכילה שלש כניסות. המפה של קובץ FILE 1 נמצאת בגוש פיסי 7, אך מאחר ולא היה מספיק מקום היא נמשכת בגוש פיסי 10. גושים פיסיים 6, 13, 4, 16, 5 מכילים את הקובץ FILE 1 לפי

הסדר הלוגי שלו. על מנת למנוע מספר גישות רב ליחידת האחסון כאשר מטפלים בקובץ FILE 1 למשל, ניתן להעתיק את המפה שלו לזיכרון מיד עם פתיחת הקובץ.



תרשים 11-11: מיפוי קבצים

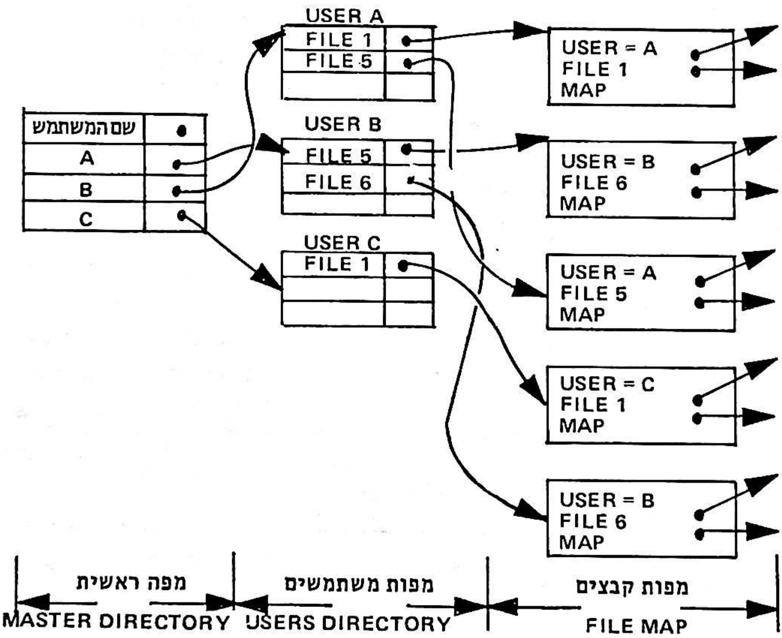
11.4 אבטחה ופרטיות בניהול קבצים

השימוש במערכות כלליות לניהול קבצים ויצירת כלים סטנדרטיים לגישה אליהם הציבה בעיה חדשה — בעיית ההגנה על הקבצים. הגנה זו מתבטאת בצורך לפקח על הגישות לקובץ והשימוש בו למטרות קריאה או גם למטרות כתיבה למשל, ומציבה מספר דרישות לפיקוח ולהגדרת סמכויות.

זיהוי הבעלים של הקובץ (File Owner)

הפניה לקבצים תוך שימוש בשם הסימלי נוחה מאד. הבעיה מתעוררת כאשר משתמשים שונים בוחרים בשם סימלי זהה לקבצים שונים. לדוגמא השם MLAY יכול לשמש משתמשים שונים לאחסון נתונים שונים. קיימות מספר שיטות לטיפול בבעיה זו ונבחר באחת מהן.

במקום להחזיק טבלה ראשית אחת לכל הקבצים ביחידת האחסון משתמשים בכמה רמות של טבלאות כפי שמוצג בתרשים 11-12: טבלה ראשונה מכילה את שמות המשתמשים והפניות לטבלאות המכילות את שמות הקבצים של כל משתמש ומהן, אל טבלאות מיפוי הקבצים.



תרשים 11-12: שיטת שיוך של בעלים לקבצים

בתרשים 11-12, שני משתמשים שונים (A ו B) משתמשים בשם סימלי זהה, FILE 5, לניהול קבצים שונים. השימוש בשתי הטבלאות מאפשר ניהול שני הקבצים ללא תלות בעובדה שיש להם שם זהה והם שייכים למשתמשים שונים. את שתי הטבלאות ניתן לנהל בדומה למפה הראשית באמצעות מפות שמאוחסנות בדיסק.

לבניית קובץ חדש נשתמש בפקודת JCL שבה מוזכר שם המשתמש, לדוגמא:

CREATE, file-name, USER = A

קשירת הקובץ לתוכנית תבוצע באמצעות פקודת JCL שגם בה מוזכר שם המשתמש:

ATTACH, file-name, USER = A

לאחר ביצוע פקודה זו, ניתן להפעיל את תוכנית היישום כדי לקרוא ולעדכן רשומות בקובץ.

קביעת זכויות הטיפול בקובץ

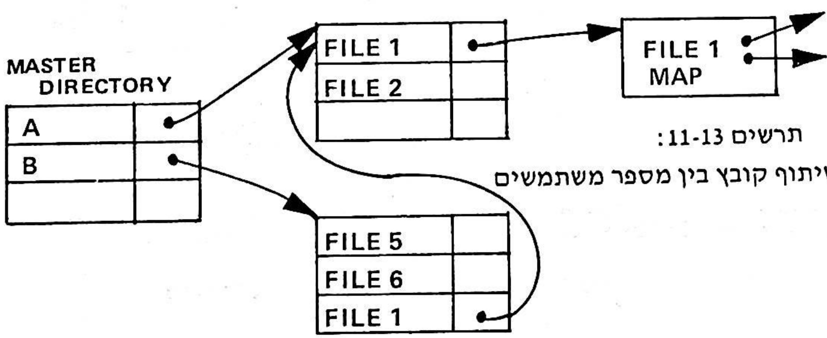
ברוב המערכות יש לאפשר למספר משתמשים שונים לגשת לאותו קובץ. על בעל הקובץ להגדיר איזה מבין הפעולות האפשריות מותר להם לעשות:

— קריאה בלבד (R),
— כתיבת רשומות חדשות (W),
— ביטול רשומות בלבד (D),
— עדכון רשומות (U),
— ביטול הקובץ (DF).

כדי להגן על הקובץ ניתנת לרבים הרשאה **לקריאה בלבד**, ורק לבעלי הקובץ, או האחראי לו ניתנת הרשאה לביצוע כל הפעולות. בעל הקובץ מגדיר את אוכלוסית המשתמשים המורשים לגשת לקובץ באמצעות פקודת JCL מיוחדת. לדוגמא הוראה המאפשרת למשתמשים A ו־B לפנות לקובץ לשם קריאה בלבד.

SHARE, file-name, USER = A, USER = B, ACCESS = R

הוראה זו תגרום לקישור טבלאות משתמשים אחדים אל קובץ אחד כפי שמוצג בתרשים 11-13.



תרשים 11-13:
שיתוף קובץ בין מספר משתמשים

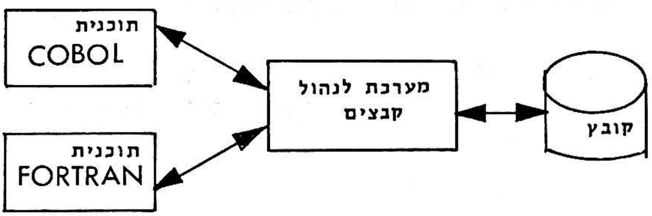
שיטת הגנה נוספת היא קביעת **סיסמא** (PASSWORD) אשר ניתנת בכרטיס הפיקוח ונבדקת בזמן OPEN. רק מי שבידיו הסיסמא יכול לגשת לקובץ. ניתן לקבוע סיסמא שונה לכל אחת מהפעולות המותרות בקובץ. את רשימת הסיסמאות ניתן לשמור בטבלת הקבצים של המשתמש בדומה לנתונים אחרים.

11.5 סיכום

לשימוש במערכות סטנדרטיות לניהול קבצים יתרונות רבים אשר נסכם אותם בקצרה. החסרון הבולט ניתן גם הוא. חסרון זה אינו מהווה כל מגבלה למשתמש הרגיל ואמנם במערכות מיוחדות נכתבות מערכות ניהול קבצים עבור המשתמש לצרכיו בלבד.

יתרונות:

א. קיימת אי־תלות בין השפה שיוצרת את הקובץ לבין השפה המשתמשת בקובץ. כל הטיפול בקובץ הוא חיצוני לתוכנית המשתמש.



תרשים 11-14: בנית קובץ בקובול ושימוש בו בפורטרן

ב. המשתמש אינו מטפל בייצוג הפיסי של הקבצים ביחידות האחסון השונות אלא בצד הלוגי של ניהול הנתונים בלבד.

ג. קיימת בקרה ועקביות בעת עיבוד הקובץ מאחר וכל הגישות מבוצעות באמצעות התוכנה הסטנדרטית.

ד. קיימת תמיכה במגוון של שיטות ארגון קבצים וסוגי רשומות.

ה. עלות נמוכה יחסית, מאחר וזוהי תוכנה סטנדרטית לכל המשתמשים ולכל הקבצים ולא פיתוח עצמאי המותאם לכל מקרה בנפרד.

חסרונות:

— מגבילה את המשתמש למספר מצומצם יחסית של שיטות ארגון קבצים.

11.6 שאלות

1. הסבר מהי טבלת תיאור קובץ (FDT). מדוע אין אפשרות למלא את תוכנה בזמן הידור? מהם היתרונות של מילוי הטבלה בזמנים שונים?
2. כיצד לדעתך ניתן לפתור את בעית ניהול הנתונים ללא מערכות מיוחדות לניהול נתונים. מה היתרונות והחסרונות של פתרון מסוג זה. תן דוגמא.
3. האם תוכנית לניהול נתונים במחשב אחד יכולה להתאים למחשב של יצרן אחר, אע"פ שהמשתמש רואה לפניו מבנים לוגיים שוים בשני המחשבים. הסבר את תשובתך ותן דוגמא.
4. הסבר מה צריכה המערכת לניהול קבצים לעשות כדי לאפשר גישה לקובץ
   — סדרתי,
   — אינדקס סדרתי,
   — ישיר.
5. מדוע יש צורך לסגור קובץ? מה יכול לקרות אם התוכניתן שוכח לסגור את הקובץ?
6. כיצד יודעת תוכנית היישום שהרשומה המבוקשת לא נמצאת בקובץ?
7. מהי טבלת VTOC? היכן היא מאוחסנת ומתי משתמשים בה?

נספח א' — ביבליוגרפיה

הרשימה הביבליוגרפית כוללת מגוון של ספרים, חוברות, תקופונים ומאמרים בנושאים שנדונו בספר זה.

בנוסף קיימת ספרות רבה בתחומים שונים המתפרסמת על ידי יצרני המחשבים ועל ידי בתי־תוכנה המפתחים מערכות תוכנה לניהול קבצים ובסיסי נתונים.

הקורא המעונין יוכל לבחור חומר נוסף הכולל-גם תיאור מערכות חומרה ויחידות איחסון ותיאור יישומים והפתרון שניתן בהם לבעית אחסון נתונים.

1. אריאל צבי: ״ארגון קבצים — מערכות 370״. י.ב.מ., 1972.
2. ארגון וניהול קבצים, המכון לפריון העבודה והייצור.
3. גולדנברג אליעזר: ״מבוא למחשבים״, מכלול, הוצאה לאור, 1975.
4. נוימן זאב, הורוביץ ישראל: ״מחשבים ומערכות מידע״, גומא, 1976.
5. עמיהוד יצחק: ״המחשב האלקטרוני ועיבוד נתונים״, הוד־עמי, 1973.
6. עמיהוד יצחק: ״תכנון ותכנות יישומי מחשבים״, הוד־עמי, 1975.
7. עמיהוד יצחק, קורפל יונתן: ״יסודות בניתוח ותכנון מערכות מידע״, הוד־עמי בשיתוף המכון לפריון העבודה והייצור, 1980.

1. *O. Amble, D.E. Knuth.* Ordered Hash Tables, Computer Journal, Vol. 17, No. 2, 1974.
2. *C.W. Bachman.* The Evolution of Storage Structures, Comm. ACM., Vol. 15, No. 7, 1972.
3. *W. Buchholz.* File Organization and Addressing. IBM System Journal, Vol. 2, No. 2, 1963.
4. *E.G. Coffman.* File Structures Using Hashing functions. Comm ACM.,Vol. 13, No. 7, 1970.
5. *G. Davis.* Computers and Information Processing. McGraw-Hill, 1978.
6. *G.G. Dodd.* Elements of Data Management Systems. In: Data Base Management. Petrocelli Books, 1974.
7. *J. Donovan.* Systems Programming. McGraw-Hill, 1972.
8. *M. Elson.* Data Structures. S.R.A, 1975.
9. *I. Flores.* Data Structure and Management. Prentice-Hall, 1970.
10. *IBM.* DL/1 DOS/VS Guide for New Users. SH24-5001-2.
11. *IBM.* IMS/VS General Information Manual. GH20-1260-7.
12. *IBM.* OS/VS VSAM Planning Guide. GC26-3749-0.
13. *J. Inglis. Inverted Indexes in Multi-List Structures. Computer Journal, Vol. 17, No. 1, 1974.*
14. *J. Inglis.* Flexibility of Blocklength for Tapes. Computer Journal, Vol. 17, No. 4, 1973.
15. *P.C. Karlton.* Performance of Height-Balanced Trees. Comm.ACM., Vol. 19, No. 1, 1976.
16. *H. Katzan.* Computer Organization and System/370. Van Nostrand Reinhold Co., 1971.
17. *H. Katzan.* Computer Data Management and Data-Base Technology. Van Nostrand Reinhold Co., 1975.
18. *G.D. Knot.* Hashing Functions and Hash-Tabels. Computer Journal, Vol. 18, No. 3.
19. *D.E. Knuth.* The Art of Computer Programming. Vol. 1: Fundamental Algoritms, 1968. Vol. 3: Sorting and Searching, 1973. Addison-Vesley Publishing Co.
20. *D. Lefkovitz.* File Structures for On-Line Systems, Hayden Book Co., 1974.
21. *A.M. Liester.* Fundamentals of Operating Systems. McGraw-Hill, 1972.
22. *K. London.* Techniques for Direct Access. Auerbach, 1972.
23. *C. Mader.* Information Systems: Technology, Economics, Applications. S.R.A., 1974.

24. *J. Martin.* Computer Data-Base Organization. Prentice-Hall, 1975.
25. *D.C. Roberts.* File Organization Techniques. In: Advances in Computers, Vol. 12, 1972. Academic Press.
26. *E.H. Sussenguth.* Use of Tree Structures for Processing Files. Comm. ACM.Vol. 6, No. 5, 1963.
27. *H. Stone.* Introduction to Data-Structures and Computer Organization. McGraw-Hill, 1974.
28. *R.E. Wagner.* Indexing Design Considerations. IBM Systems Journal, Vol. 12, No. 4, 1973.
29. *S.J. Waters.* File Design Fallacies. Computer Journal, Vol. 15, No. 1, 1972.
30. *S.J. Waters.* Estimating Magnetic Disk Seeks. Computer Journal, Vol. 18, No. 1, 1975.
31. *G. Wiederhold.* Database Design. McGraw-Hill.
32. *E. Yourdon.* Design of On-Line Computer Systems. Prentice-Hall, 1972.

נספח ב' — מילון מונחים

מילון המונחים הקצר המצורף מגיש לקורא את המונחים העיקריים שהוא נזקק להם בעבודתו בנושא שבו דן הספר. הקבלת המונחים באנגלית ובעברית תסייע לקורא בעת עיון בספרות העזר הענפה שרובה נכתבה באנגלית.

Access	גישה
Access Method	שיטת גישה
Access Time	זמן גישה
Activity Rate	שעור פעילות
Address	כתובת
Address Calculation	חישוב כתובת
Algorithm	אלגוריתם
Alphanumeric Data	נתונים אלפא נומריים
Application Program	תוכנית יישום
Assembler Program	תוכנית בשפת סף
Auxilary Storage	איחסון חיצוני
Backup File	קובץ גיבוי
Balanced Tree	עץ מאוזן
Batch Processing	עיבוד באצווה
Binary	בינרי
Binary Search	חיפוש-בינרי
Binary Tree	עץ בינרי
Bit	סיבית (סיפרה בינרית)
Block	גוש
Block Length	אורך הגוש
Blocking Factor	מספר הרשומות בגוש
Buffer	מאגר
Byte	בית (8 סיביות)
Byte Per Inch (BPI)	מספר בתים לאינץ (צפיפות מגנוט)
Byte Per Second (B	מספר בתים לשניה (מהירות העברה)
Chain	שרשרת
Chain Length	אורך השרשרת
Channel	ערוץ (אפיק)
Character	תו
Check Digit	סיפרת ביקורת
Check Point	מיבדק
CPU	יחידת עבוד מרכזית (יעמ)
Collision	התנגשות
Compiler	מהדר
Cylinder	גליל (צלינדר)
Data	נתונים
Data Base	בסיס נתונים

Data Base Management System (DBMS)	מערכת לניהול בסיס נתונים
Data Description Language	שפה לתאור נתונים
Data Management	אי תלות נתונים
Data Manipulation Language	שפה לטיפול בנתונים
Data Model	מודל נתונים
Data Processing	עיבוד נתונים
Data Structure	מבנה נתונים
Density	צפיפות
Direct Access	גישה ישירה
Direct Access Storage Device (DASD)	יחידת איחסון בגישה ישירה
Directory	מדריך
Disk	דיסק
Disk Drive	כונן דיסק
Duplicate Key	מפתח כפול
Exhaustive Index	אינדקס ממצה
Field	שדה
File	קובץ
File Management System	מערכת לניהול קבצים
File Organization	ארגון הקובץ
Fixed Head Disk	דיסק עם ראשים קבועים
Floppy Disk	דיסקט
Free Space	שטח פנוי
Free Space Management	ניהול השטח הפנוי
Gap	מרווח
Generic Key	מפתח חלקי
Global Overflow Area	שטח גלישה כללי
Hardware	חומרה
Hashing	פיזור (רשומות בקובץ)
Hierarchical File	קובץ היררכי
Historical File	קובץ היסטורי
Home Address	כתובת מקורית (של הרשומה)
Input/Output (I/O)	קלט/פלט
Input/Output Channel	ערוץ קלט/פלט
Index	אינדקס, מדריך
Index Sequential	אינדקס סדרתי
Index Sequential Access Method (ISAM)	ארגון אינדקס סדרתי
Information	מידע

Information System	מערכת מידע
Interface	מימשק
Inverted File	קובץ הפוך
Inquiry	שאילתא
Job Control Language)JCL)	שפת פיקוח
Key	מפתח
Key Field	שדה המפתח
Key-To-Address Transformation	תרגום מפתח לכתובת
Label	תווית (של קובץ)
Latency	השהייה סיבובית (בדיסק)
Length Count	מונה אורך
List	רשימה
Local Overflow	גלישה מקומית
Logical	לוגי
Logical Record	רשומה לוגית
Machine Language	שפת מכונה
Magnetic Disk	דיסק מגנטי
Magnetic Tape	סרט מגנטי
Major Key	מפתח מוביל
Map	מפה
Mapping	מיפוי
Master Directory	מדריך ראשי
Master File	קובץ אב
Model	מודל, דגם
Multilist Organization	ארגון רב־רשימתי
Multiple Key Retrieval	שליפה לפי מספר מפתחות
Network	רשת
Node	צומת
Numeric Data	נתונים מספריים
Off-Line	מופרד
On-Line	מקוון
On-Line Processing	עיבוד מקוון
Operating System	מערכת הפעלה
Overflow	גלישה
Overflow Area	שטח גלישה
Output Buffer	מאגר פלט

Parity Bit	סיבית ביקורת הזוגיות
Performance	ביצוע
Peripheral Equipment	ציוד היקפי
Physical	פיסי
Physical File	קובץ פיסי
Physical Record	רשומה פיסית
Pointer	מצביע
Primary Key	מפתח ראשי, עיקרי
Primary Storage	אחסון ראשי, עיקרי
Program	תוכנית
Random Access	גישה אקראית
Random Access Storage	איחסון בגישה אקראית
Random File Organization	קובץ מאורגן אקראית
Read/Write Head	ראש קריאה/כתיבה
Real Time	זמן אמת
Record	רשומה
Record Length	אורך רשומה
Record Key	מפתח הרשומה
Recovery	התאוששות
Reorganization	ארגון מחדש (של קובץ)
Response Time	זמן תגובה
Ring Structure	מבנה טבעתי
Root Record	רשומת שורש (במבנה הירכי)
Search Key	מפתח חיפוש
Secondary Index	אינדקס משני
Secondary Key	מפתח משני
Secondary Storage	איחסון משני
Seek	חיפוש
Seek Time	זמן חיפוש
Segment	סגמנט
Selective Index	אינדקס סלקטיבי
Sequential	סדרתי
Sequential Processing	עיבוד סדרתי
Sequential Access Method (SAM)	ארגון סדרתי
Software	תוכנה
Software Package	חבילת תוכנה
Sort	מיון
Standard	תקן

Start/Stop Time	זמן התחלה/עצירה
Storage	אחסון
System	מערכת
System Analysis	ניתוח מערכת
System Developement	פיתוח מערכת
Table	טבלה
Tape Drive	כונן סרטים
Tele Processing	עיבוד רחק
Trailer	נגררת
Track	מסילה
Track Index	אינדקס מסילות
Transaction File	קובץ תנועות
Transfer Rate	מהירות העברה
Tree Index	אינדקס במבנה עץ
Tree Structure	מבנה עץ
Update	עדכון
User	משתמש
Utility Program	תוכנית שירות
Virtual Storage	זכרון בפועל
Word	מילה
Working Storage	אזור עבודה
Write	כתיבה (של רשומה לקובץ)

רשימות

רשימות

— מתי ואיך יש למידע ערך?
— מהם מבני נתונים, אינדקסים, טבלאות, שרשרות ועצים?
— מהו ניהול נתונים?
— מהם ההבדלים בין ייצוג לוגי וייצוג פיסי, בין יחידות אחסון שונות, ועוד...
— מהם שיקולי התכנון של קבצים?

ארגון נתונים וניהול קבצים נועד להכשיר מנתחי מערכות, מתכננים ומתכנתי מערכות העוסקים בפיתוח תוכנה ואחראים לניהול נתונים (Data Administrators).

הספר מותאם לתוכניות הלימודים המופעלות באוניברסיטאות, בבתי ספר מקצועיים, ובקורסים נבחרים למנתחי מערכות. בסיום כל פרק הובאו שאלות לחזרה ולתרגול.

על המחבר

הייפרמן רז, יליד 1949, תואר ראשון בסטטיסטיקה וכלכלה ותואר שני בחקר ביצועים מטעם האוניברסיטה העברית בירושלים. בוגר קורס מנתחי מערכות של המכון לפריון העבודה והייצור.
עובד בחברת „טכנולוגיה מתקדמת" כמנתח מערכות. עד שנת 1981 עבד במרכז שרותי המידע של משרד התקשורת בתפקידים של מנתח מערכות ומנהל תחום מערכות לוגיסטיות.
מרצה במדור ללימודי חוץ של הטכניון בנושאי ארגון קבצים, מבנה נתונים ומערכות לניהול בסיסי נתונים. הרצה במספר רב של קורסים בנושאים אלה מטעם משרד התקשורת, המכון לפריון העבודה והייצור ואמק"ד (ארגון משתמשי ציוד קונטרול דאטה).

ספרים נוספים בהוצאתנו

1. המחשב האלקטרוני ועיבוד הנתונים, י' עמיהוד
2.* תכנון ותכנות יישומי מחשבים, י' עמיהוד
3.* הקלט למחשב, ס' מצפי
4.* יסודות בניתוח ותכנון מערכות מידע, י' עמיהוד ו־י' קורפל
5.* מערכות תקשורת נתונים, א' סקופ

* באישור משרד החינוך